VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODUL PRO SPRÁVU ÚKOLŮ DO SYSTÉMU NLPIS

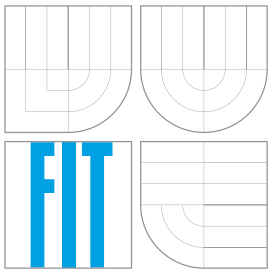
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ ŽUREK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODUL PRO SPRÁVU ÚKOLŮ DO SYSTÉMU NLPIS

MODULE FOR TASK ADMINISTRATION FOR NLPIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ ŽUREK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV DYTRYCH

BRNO 2012

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2011/2012

Zadání bakalářské práce

Řešitel: **Žurek Aleš**
Obor: Informační technologie
Téma: **Modul pro správu úkolů do systému NLPIS**
Module for Task Administration for NLPIS
Kategorie: Web

Pokyny:

1. Seznamte se s jazyky a prostředky pro tvorbu webových informačních systémů
2. Prostudujte informační systém skupiny zpracování přirozeného jazyka a potřeby jeho uživatelů
3. Navrhněte modul pro správu úkolů v informačním systému NLPIS tak, aby bylo možné kontrolovat stav jednotlivých úkolů. Modul umožní jednoduchou správu úkolů a poskytne různé pohledy na úkoly.
4. Implementujte navržené řešení
5. Zhodnoťte dosažené výsledky, srovnejte zvolené přístupy s alternativními metodami a navrhněte další možná rozšíření a vylepšení do budoucna

Literatura:

- podle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a 3

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dytrych Jaroslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2011
Datum odevzdání: 16. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce analyzuje dosavadní informační systém NLPIS a jeho součásti. Obsahuje návrh a implementaci, která rozšiřuje dosavadní možnosti systému NLPIS a upravuje jeho součásti. Zaměřuje se přitom na správu úkolů a na propojení systému NLPIS s MediaWiki. Součástí řešení je i rozbor nově přidaných tříd.

Abstract

This bachelor thesis analyzes the current information system NLPIS and its components. It include design and implementation that extends the capabilities of existing system NLPIS and adjusts its components. It aims to manage tasks and to link the system NLPIS with MediaWiki. The solution includes an analysis of the newly added classes.

Klíčová slova

NLPIS, informační systém, správa, úkoly, PHP, třída, MySQL, databáze, MediaWiki

Keywords

NLPIS, information system, administration, task, PHP, class, MySQL, database, MediaWiki

Citace

Aleš Žurek: Modul pro správu úkolů do systému NLPIS, bakalářská práce, Brno, FIT VUT v Brně, 2012

Modul pro správu úkolů do systému NLPIS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Aleš Žurek
15. května 2012

Poděkování

Především bych chtěl poděkovat svému vedoucímu bakalářské práce panu Ing. Jaroslavu Dytrychovi, který mi velmi pomohl, poskytoval cenné rady a aktivně odpovídal na dotazy. Dále bych také chtěl poděkovat své rodině a přátelům, kteří mě v této bakalářské práci podporovali.

Obsah

Kapitola 1

Úvod

Cílem této práce bylo implementovat správu úkolů do informačního systému NLPIS a do projektů k němu přidružených. Hlavním přidruženým projektem je diplomová práce pána Ing. Aleše Vavřínka s názvem Automatické generování wiki stránek z e-mailů a IS [22], kterou bylo potřeba upravit a doplnit o novou implementaci úkolů. K úpravě správy úkolů patří také přidání automatické kontroly úkolů pro nové studenty, které se nachází na stránce MediaWiki.

Požadavky kladené na implementaci jsou uvedeny v kapitole č. 2. Ta je rozdělena na podkapitoly, které se podrobněji zabývají požadavky kladenými na bakalářskou práci.

Po provedení analýzy byly vybrány technologie, které budou použity pro implementaci. Tyto technologie jsou popsány v kapitole č. 3.

Implementaci předcházel detailní návrh řešení, který obsahuje přehled změn, které bylo třeba provést, a návrh nových tříd. Návrh řešení je popsán v kapitole č. 4.

Popis implementační části je uveden v kapitole č. 5. Ta je rozdělena na podkapitoly, které jsou určeny jednotlivým částem bakalářské práce. V této kapitole jsou také uvedeny problémy, které byly v průběhu implementace objeveny.

Po dokončení implementace došlo k testování, které je popsáno v kapitole č. 6.

Na závěr jsou v kapitole č. 7 shrnuty provedené změny v systému NLPIS a jeho součástech. Dále je v této kapitole rozepsáno možné další využití tříd pro správu úkolů.

Obsahem této práce je i příloha, která obsahuje diagramy tříd, vytvořených pro správu úkolů systému NLPIS.

Kapitola 2

Analýza

Tuto bakalářskou práci je možné rozdělit na tři části, které jsou relativně málo provázané. Po definici vhodných rozhraní bude možné tyto části navrhnout a implementovat jednotlivě a následně integrovat do existujícího systému.

První část bakalářské práce bude úvodní, na které si vyzkouším práci se základy systému MediaWiki a systému NLPIS. V této části se budu zabývat úpravou diplomové práce Ing. Aleše Vavřínka, která nese název Automatické generování wiki stránek z e-mailů a IS.

Druhá část bakalářské práce je vytvořit správu úkolů v informačním systému NLPIS, která bude umožňovat zaznamenávat historii stavů úkolů a historii změn úkolů.

Závěrečnou částí bakalářské práce je vytvořit skript, který bude zpracovávat a kontrolovat stránku v MediaWiki, na které se přihlašují studenti k vypsaným úkolům.

2.1 Automatické generování wiki stránek z e-mailů a IS

Úkolem je změnit kód hotové diplomové práce Ing. Aleše Vavřínka. Nedostatkem této diplomové práce je, že se generuje hlavička vždy pro všechny projekty a všechny skupiny v systému. Tedy uživatel si nemůže libovolně upravit vygenerovanou wiki stránku dle svého uvážení, aniž by mu jeho část, kterou editoval, nebyla přemazána či přesunuta. V této generované hlavičce jsou vypsány základní údaje o projektech a skupinách jako je název, vedoucí apod.

Je kladen požadavek, aby bylo možné vybírat projekty a skupiny, ke kterým se bude automaticky generovat hlavička na stránkách MediaWiki. Jelikož se na stránce může vyskytovat více projektů, tak pro snadnější aplikaci pravidel, které z projektů se nemají generovat automaticky, existují dva způsoby, jak zamezit, aby systém projektu automaticky generoval svoji hlavičku do MediaWiki stránky.

Prvním způsobem je zabránění generování specifického projektu, ale tento způsob je nevyhovující, pokud se na jedné wiki stránce nachází více projektů a chceme, aby stránka nebyla automaticky modifikována. Druhý způsob je tedy zamezení automatického generování wiki stránky pro zadanou stránku, která je definována pomocí URL adresy. Pro zamezení generování specifické skupiny lze použít obě metody, protože na jedné MediaWiki stránce se smí vyskytovat pouze jedna skupina.

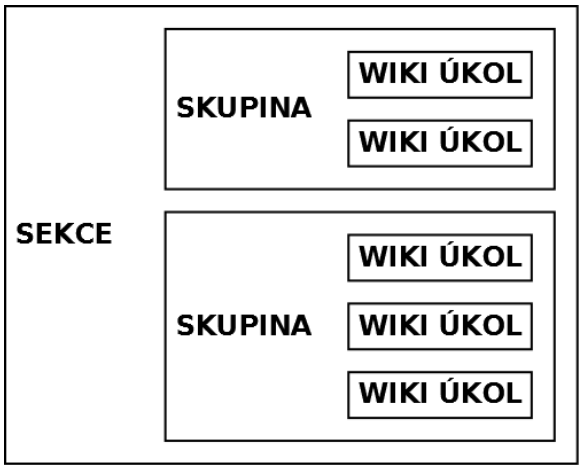
Výběr projektů a skupin, které nemají být vygenerovány, se může provádět buď přímo v systému MediaWiki, a nebo v informačním systému NLPIS. Pro systém MediaWiki je tedy nutné vytvořit označení, které se vloží k projektu nebo skupině, a tímto bude projekt nebo skupina označena, že se nemá generovat.

2.2 Správa úkolů v systému NLPIS

Při zpracování této části bakalářské práce je zapotřebí nastudovat informační systém NLPIS. Jelikož v tomto informačním systému jsou úkoly k projektům zadány pouze pomocí textového pole, není možné nijak kontrolovat termíny a míru splnění jednotlivých úkolů řešitelů. Vedoucí projektu tedy nemá představu, na kolik procent jsou jednotlivé úkoly projektu splněny a jestli řešitel daný úkol vlastně řeší.

Cílem je vytvořit správu úkolů, která bude shromažďovat historii jednotlivých úkolů a zároveň jednotlivé úkoly budou procházet předdefinovanými stavy, které budou také ukládány. Toto rozšíření s sebou přinese změny v různých částech systému NLPIS. Musí se změnit stránky, na kterých se tyto informace vypisují, bude potřeba pozměnit dotazy na databázi a změny se budou týkat i diplomové práce pána Ing. Aleše Vavřínka, kde se také vypisují úkoly zadané jednotlivým řešitelům.

Očekávaný výsledek této práce je výpis veškerých změn úkolů a ke každému úkolu budou vypsány stavy, kterými daný úkol procházel. Důraz je kladen i na rychlost prováděných operací, protože stránka s úkoly v informačním systému NLPIS zobrazuje všechny úkoly, které se v databázi nacházejí, a při špatném návrhu řešení by došlo k neúnosně dlouhé načítací době stránky systému, což je nepřijatelné.



Obrázek 2.1: Struktura stránky MediaWiki pro přihlašování k úkolům pro nové studenty

2.3 Přihlašování k úkolům na stránce MediaWiki

Tato část bakalářské práce je zaměřena na práci se stránkou v systému MediaWiki, kde se přihlašují studenti na úkoly vypsané výzkumnou skupinou NLP. Stránka má být v určitých časových intervalech kontrolována skriptem a změny se na této stránce musí projevit v databázi informačního systému NLPIS.

Zároveň lze provádět změny v informačním systému NLPIS a takto provedené změny musí být viditelné na stránce v systému MediaWiki. Proto se stránka v MediaWiki musí zároveň chovat nejen jako výstup dat z databáze, ale i jako vstupní zdroj dat. Při neshodě dat v databázi a na stránce MediaWiki bude potřeba určit, který ze zdrojů dat obsahuje platné informace a jaký zdroj informací má být aktualizován. Pokud dojde ke změně

dat v databázi a na stránce MediaWiki, tak přednost dostane MediaWiki a data budou v databázi přepsána. V informačním systému nemůžeme úkoly přímo smazat, protože stránka se aktualizuje v určitých intervalech a mohlo by dojít k situaci, kdy po smazání úkolu v databázi se přihlásí na tento úkol řešitel. Taková situace musí být patřičně ošetřena.

Jak je vidět na obrázku č. 2.1, tak stránka obsahuje jednu sekci, která se skládá z titulku a textu, který danou sekci popisuje. Tato sekce neobsahuje úkoly, ke kterým by se studenti mohli přihlásit. Dále tato sekce obsahuje skupiny, které mají svůj název a popis. Tyto skupiny již v sobě mohou sdružovat úkoly, na které se studenti mohou přihlašovat. Nejkomplikovanější část stránky jsou úkoly, protože obsahují nejvíce informací. Úkol se skládá z nadpisu, za kterým se mohou vyskytovat v závorce vypsaní řešitelé přihlášení k tomuto úkolu. Dále následuje text označující vedoucího a za ním jméno a e-mail vedoucího úkolu. Za tímto jménem může být uvedeno ještě jedno jméno a e-mail, které udává příjemce zpráv pro tento wiki úkol. Po vedoucím může úkol obsahovat nepovinnou položku projekt, kde je uveden název skupiny projektů, pod kterou wiki úkol spadá. Poté následuje textový popis úkolu, který je na začátku označen znakem hvězdičky (syntaxe jazyka MediaWiki pro položku seznamu).

Kapitola 3

Použité technologie

Informační systém NLPIS pracuje s technologií PHP a využívá MySQL databázi. Využití těchto technologií je potřeba pro změnu aktuálního řešení a pro přidání požadovaných změn. Pro dynamický formulář budu volit skriptovací jazyk JavaScript, protože je již zaveden na některých stránkách systému NLPIS. Jazyky HTML a CSS budou využity při formátování kódu, který se bude vkládat do systému NLPIS.

3.1 PHP

Zkratka PHP znamená "PHP: Hypertext Preprocessor" (jedná se o rekurzivní zkratku[1]) [16]. Autorem jazyka PHP je Rasmus Lerdorf, který tento jazyk, spolu s ostatními spolupracovníky, dále rozšiřuje [15]. PHP je skriptovací jazyk běžící na straně serveru, díky kterému mohou být webové stránky opravdu dynamické [4]. Najde uplatnění také při generování obsahu XML dokumentů.

Aktuální verze PHP je 5.4.1, ale jelikož na serveru, kam bude tato bakalářská práce nasazena, je nainstalována verze PHP 5.2.6, tak z důvodu lepší kompatibility byla zvolena tato verze.

3.2 MySQL

Význam názvu MySQL je My Structured Query Language, kde My je jméno dcery hlavního autora MySQL a spoluzakladatele MySQL AB, kterým je Ulf Michael Widenius [13]. SQL je dotazovací jazyk, který slouží pro komunikaci s databází. Architektura MySQL je velmi odlišná od ostatních databázových serverů a díky tomu je užitečná pro široké spektrum použití. MySQL není dokonalá, ale je dostatečně flexibilní, aby pracovala v náročných prostředích, jako jsou webové aplikace [26]. Pomocí tohoto jazyka vytváříme a měníme strukturu databáze, vkládáme, editujeme a čteme požadovaná data.

Aktuální verze MySQL je 5.5.23, kterou dále vyvíjí firma MySQL AB. Kvůli implementaci na server, kde bude tato bakalářská práce fungovat, jsem pracoval s verzí MySQL 5.0.51a.

Další očekávaná verze MySQL je 5.6.5, která má přinést řadu vylepšení výkonu a spolehlivosti databáze [14].

[1] Rekurzivní zkratka, obsahuje ve svém názvu vlastní zkratku [1].

3.3 HTML

Celý název HTML zní HyperText Markup Language [18]. HTML je značkovací jazyk definující syntaxi a umístění speciálních vložených pokynů, které nejsou zobrazeny v prohlížeči, ale doporučí mu, jak zobrazit obsah dokumentu, včetně textu, obrázků a dalších pomocných médií [9]. Tvůrce tohoto značkovacího jazyka je Tim Berners-Lee [19]. Nyní je jazyk HTML vyvíjen a rozšiřován mezinárodním konsorciem W3C.

Aktuální verze HTML je 4.01. Při tvoření HTML kódu je dodržena tato specifikace, protože je použita i v celém systému NLPIS.

Momentálně se pracuje na verzi HTML 5, která je již na webových stránkách používána, ale oficiální status HTML5 ke dni 22.4.2012 stále zní pracovní návrh [5].

3.4 CSS

CSS celým názvem Cascading Style Sheet, se používá k stylování vzhledu HTML značek v dokumentu. S CSS, je možné zcela změnit způsob, jakým jsou prvky prezentovány pomocí uživatelského agenta (webového prohlížeče) [11]. Tvůrcem kaskádových stylů je Håkon Wium Lie, který spolupracoval s dalšími vývojáři, mezi které patří Tim Berners-Lee a Robert Cailliau [24].

Aktuální vývoj opět spadá pod konsorcium W3C, kde pracují na CSS verze 3., které je již na webových stránkách používáno, ale některé jeho součásti se ke dni 22.4.2012 stále nacházejí ve stavu pracovního návrh. Tedy oficiálně vydaná aktuální verze kaskádových stylů je verze CSS 2.1. Tato verze je také využita na stránkách informačního systému NLPIS.

3.5 JavaScript

Autorem JavaScriptu je jmenuje Brendan Eich. JavaScript je skriptovací jazyk, který umožňuje vylepšit statickou webovou aplikaci tím, že poskytne dynamický, individualizovaný a interaktivní obsah [25]. JavaScript je standardizován ECMAscriptem, jehož aktuální edice nese označení 5.1. Aktuálně je vyvíjen pod záštitou Mozilly, kde momentálně Brendan Eich působí [12].

3.6 jQuery

JQuery je relativně nová technologie, kterou vyvinul John Resig [8]. Jedná se o JavaScriptový framework. Ten umožňuje měnit obsah HTML dokumentů pomocí manipulace s modelem (DOM), který prohlížeč vytváří při zpracování HTML stránky [2].

Nejnovější verze jQuery je 1.7.2. V této bakalářské práci je využito jQuery knihovny, která se již nacházela v informačním systému NLPIS, a to verzi 1.7.1. Aktuální vývoj frameworku stále pokračuje pod vedením jeho autora.

3.7 DOM

Zkratka DOM znamená Document Object Model. Jedná se o API, které se využívá pro dynamický přístup k obsahu HTML a XML dokumentů, díky kterému lze upravovat strukturu a obsah HTML a XML dokumentu. DOM je často nesprávně označován za nějaký druh

skriptování pro webové stránky, ale čisté DOM skriptování zahrnuje pouze ty funkce a metody začleněné do W3C DOM specifikace, to znamená, žádné proprietární funkce prohlížeče [21]. DOM byl vyvinut v rámci konsorcia W3C, aby bylo docíleno sjednocení přístupu k prvkům HTML a XML dokumentů [7].

Aktuální verze je DOM Level 3. Vývojová skupina, která měla na starosti DOM, již ukončila svou činnost [6].

3.8 XML

XML neboli eXtensible Markup Language byl vytvořen skupinou XML Working Group v rámci konsorcia W3C. XML je nástroj pro ukládání dat, konfigurovatelný pro jakýkoliv druh informací, vyvíjený a otevřený standard přijatý každým, od bankéřů až po webmastery [20]. Jedná se o aktuální verzi 1.1 a vývoj dále pokračuje. Aktuálně na XML pracuje několik týmů z konsorcia W3C [23]. V rámci této bakalářské práce je využit jazyk XML verze 1.0, pomocí kterého systém MediaWiki odpovídá na zaslané požadavky.

3.9 MediaWiki

Autor MediaWiki se jmenuje Lee Daniel Crocker [10]. MediaWiki slouží k hromadné správě článků, které mohou uživatelé upravovat a rozšiřovat.

Aktuální verze MediaWiki je 1.19.0, ale na serveru Merlin, na kterém je stránka, se kterou budu v rámci své bakalářské práce pracovat, umístěna, je nainstalovaná verze 1.16.5.

Kapitola 4

Návrh řešení

4.1 Automatické generování wiki stránek z e-mailů a IS

V diplomové práci pána Ing. Aleše Vavřínka je zapotřebí provést změnu, která nebyla původně v plánu zmíněné diplomové práce. Touto změnou je zamezení automatického generování obsahu wiki stránky v systému MediaWiki pro vybrané projekty a skupiny. Jelikož pán Ing. Aleš Vavřínek takovou úpravu nepředpokládal, bude zapotřebí změnit kód této práce na více místech, protože v diplomové práci se částečně spoléhá na to, že budou vždy vygenerovány všechny projekty a skupiny. Největší problém se vyskytuje na stránce projektů, protože na jedné stránce se může vyskytovat více projektů, u kterých je potřeba zjistit, zda se mají zkontrolovat a vygenerovat.

Původně bylo cílem změnit SQL dotazy v souboru, který zajišťuje automatické generování wiki stránek z e-mailů a informačního systému, aby pracovaly pouze se skupinami a projekty, které se mají generovat. Toto řešení se zdálo být jako správné, ale bohužel se ukázalo jako nefunkční na stránkách, kde se vyskytovalo více projektů. Pokud na stránce existoval projekt, u kterého se měla generovat hlavička, tak byla poté přegenerována celá wiki stránka včetně všech projektů, které se na stránce vyskytovaly. Proto bylo potřeba začít analyzovat jednotlivé funkce pána Ing. Aleše Vavřínka a zajistit negenerování hlaviček projektů a skupin na nižší úrovni.

Nakonec byla zvolena možnost, kdy jsem si vytvořil vlastní SQL dotaz, který zjistí, jestli se mají na stránce generovat hlavičky projektů a skupin. Pokud se stránka nemá generovat vůbec, tak bude funkce ukončena a následně bude načtena další stránka ke zpracování. Když neexistuje zákaz generování hlavičky pro celou stránku, tak se dále u každého projektu nebo skupiny (skupina může být pouze jedna), které se vyskytují na stránce, zjistí, jestli se má generovat hlavička a pokud nemá, tak se daný projekt nebo skupina přeskočí. U projektů to znamená, že se buď bude kontrolovat další projekt, nebo se ukončí generování aktuální stránky, zatímco u skupiny může dojít pouze k ukončení generování stránky.

V diplomové práci pána Ing. Aleše Vavřínka bylo na stránkách s více projekty zjištěno zjednodušení, které vycházelo z toho, že projekty, které mají být na stejné stránce, budou vždy generovány všechny a navíc, že vždy musí z databáze přijít ve stejném pořadí. Budu muset tedy upravit logiku kontroly a generování obsahu stránky. Provede se vkládání identifikátorů v rámci HTML komentáře. Díky těmto identifikátorů víme, které projekty jsou na stránce a v jakém pořadí se nacházejí. Zásluhou této informace můžeme předat v upravovaném kódu správný text projektu, který se má zkontrolovat a případně znovu vygenerovat.

Aby bylo možné mimo informační systém měnit zákaz generování hlavičky, musí existo-

vat určité řetězce, které když budou při průchodu stránky nalezeny, tak zakážou generování hlavičky. Zároveň pokud má stránka zákaz generování a tento speciální řetězec bude odstraněn, tak se má v databázi uložit hodnota, aby bylo možné znova generovat hlavičku projektu nebo skupiny. Z tohoto důvodu nemůže být zavedena ani kontrola zákazu generování hlaviček pro celou stránku dříve, protože by se neodhalilo odstranění této značky ze stránky. Proto bude nutné postupně procházet všechny jednotlivé stránky a hledat na nich, jestli nedošlo k odstranění onoho speciálního řetězce. Jelikož nechceme, aby byl tento speciální řetězec vidět na MediaWiki stránce, tak jedinou možností je obalit jej do HTML komentáře. Díky tomu zůstává tento řetězec skrytý před návštěvníky a zobrazen bude pouze při editaci. Tyto zmiňované speciální řetězce jsou celkem tři.

Řetězce pro zakázání generování hlavičky:

- `<!- NEGENEROVAT ->` - zákaz generování hlaviček pro celou stránku
- `<!- NEGENEROVAT PROJEKT ->` - zákaz generování hlavičky pro projekt
- `<!- NEGENEROVAT SKUPINU ->` - zákaz generování hlavičky pro skupinu

Řetězce označující zákaz generování hlaviček pro projekt a skupinu mohou být zastoupeny stejným řetězcem, protože skupina a projekt nemohou být na stejné stránce.

Aby bylo možné v informačním systému NLPIS zakázat generování hlavičky projektu nebo skupiny, je třeba přidat tabulkám `projekt` a `skupina_proj` atribut `generovani_hlavicky`, který určí, zdali se má hlavička projektu, respektive hlavička stránky, kontrolovat a generovat. Tato úprava je viditelná na obrázku č. 4.1. Pro zakázaní generování stránek bude přidána tabulka `wiki_vyjimky`, která bude obsahovat URL adresy zakázaných stránek.

Protože bude možno zakázat generování hlavičky projektu nebo skupiny jak v systému MediaWiki, tak i v informačním systému NLPIS, tak bude zapotřebí určit správný zdroj dat, aby nedošlo k možné ztrátě této informace. Z tohoto důvodu bude v databázi potřeba rozlišit několik stavů výše zmíněného atributu generování hlavičky, které budou informovat o tom, v jakém stavu se generování hlavičky projektu nebo skupiny nachází. Dle tohoto stavu bude určeno, který ze zdrojů dat bude mít při zpracování dat přednost. Ale i přesto může nastat situace, kdy nebude jasné, který ze zdrojů dat je aktuální. Tato situace může nastat, když jsou zároveň změněna data jak v databázi, tak na stránce MediaWiki. Pokud toto nastane, tak je určeno, že přednost bude mít vždy stránka MediaWiki. V tomto případě tedy bude databáze přepsána daty, které se nachází na MediaWiki stránce.

4.2 Správa úkolů v systému NLPIS

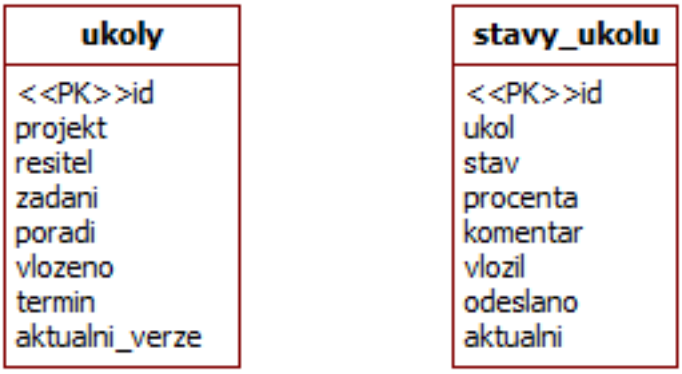
Tato část bakalářské práce zahrnuje rozšíření informačního systému NLPIS. Pro tento účel je potřeba změnit, respektive rozšířit, stávající databázi informačního systému NLPIS. Na obrázku č. 4.2 jsou vidět databázové tabulky, které jsem přidal v rámci této části bakalářské práce.

Zadání úkolů pro řešitele se může skládat z 1-N úkolů, takže bude potřeba přidat tabulku, která bude shromažďovat data o jednotlivých úkolech. V této tabulce budou uložena data, která jsou pevně spjata s úkolem. Jedná se o atributy `zadani`, `poradi` a `termin`. Tyto atributy bude uživatel měnit ve formuláři na stránce a při změně některého z těchto atributů bude vygenerován nový úkol. Kvůli požadavku udržovat historii úkolů nemůžeme starý úkol smazat, ale zároveň potřebujeme rozlišit, zda-li je úkol v databázi aktuální a k jakému úkolu

Obrázek 4.1: Databázové tabulky ve kterých se nastavuje zamezení generování hlaviček

se vztahuje. K tomuto účelu bude v databázi atribut `aktualni_verze`. Pokud se úkol změní a bude vytvořen úkol nový, tak zde bude uložen identifikátor nově vytvořeného úkolu. Takto by se docílilo provázání mezi úkoly, kdy bychom se při každém SQL dotazu dostali k stále novější verzi úkolu, ale toto řešení by bylo komplikované jak na zpracování, tak na databázi, na kterou bychom se neustále dotazovali. Proto bude můj návrh pracovat tak, že všechny starší verze úkolu budou mít v atributu `aktualni_verze` uložen pouze identifikátor nejnovějšího úkolu (při přidání novějšího úkolu se atribut `aktualni_verze` pozmění ve všech předešlých verzích úkolu). Po této úpravě lze získat pomocí jednoho SQL dotazu všechny požadované úkoly a pro správné seřazení úkolu dle jejich historie postačí seřadit úkoly dle atributu `vlozeno`, který bude obsahovat datum a čas vložení úkolu. V tabulce projekt budou samozřejmě i další atributy, které slouží pro práci s daným úkolem a umožní napojení na další tabulky v databázi.

Každý z úkolů projektu prochází určitými stavy, jenž je potřeba uchovávat z důvodu historie. Proto bude existovat tabulka `stavy_ukolu`, která bude uchovávat data, která lze změnit, aniž by došlo ke změně úkolu. Těmito daty budou atributy `stav`, `procenta` a `komentar`. Atribut `procenta` bude označovat procentuální splnění úkolu a atribut `komentar` bude sloužit k přidání komentáře k novému stavu úkolu. Pokud dojde ke změně libovolného atributu, tak bude vytvořen nový stav úkolu. Obdobně jako u úkolů bude potřeba uchovávat historii, takže starý stav nebudeme moci smazat, ale jeho atribut `aktualni` mu bude změněn tak, aby bylo možné poznat, že se již nejedná o aktuální stav úkolu. Řazení stavů úkolu se provádí stejně jako u řazení úkolů. Pro získání seřazených stavů úkolů se stavy seřadí podle atributu `odeslano`. Protože stavy úkolů může měnit vedoucí i řešitel, bude potřeba v tabulce uchovávat i identifikátor osoby, která změnu stavu provedla. Tabulka `stavy_ukolu` bude



Obrázek 4.2: Databázové tabulky vytvořené pro správu úkolů

obsahovat další atributy, které budou potřebné pro napojení na další tabulky v databázi.

4.2.1 Třídy pro správu úkolů

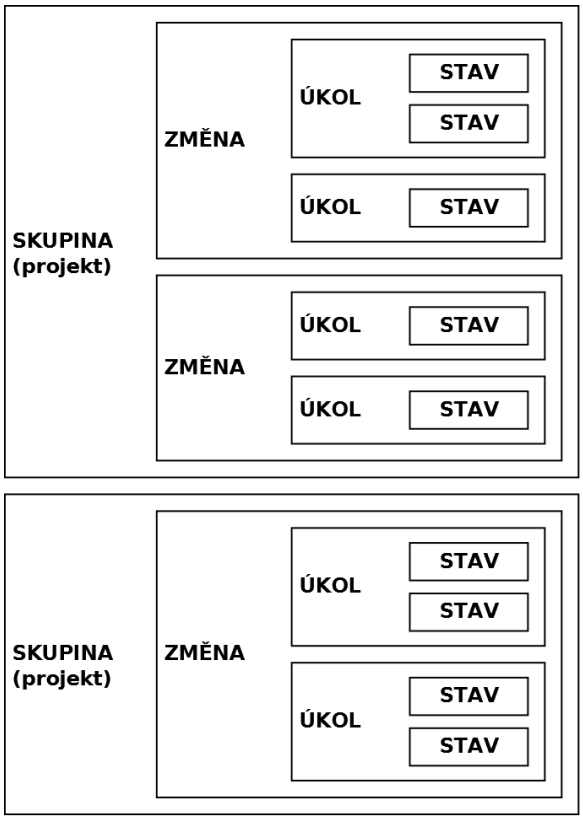
Nově vytvořený kód pro správu úkolů v systému NLPIS bude objektově orientovaný, tedy bude se skládat ze tříd - konkrétně dvou. Tyto třídy jsou zobrazeny na obrázcích v příloze B. Třída `task` bude sloužit pouze pro načtení dat z databáze a následné setřídění dat a jejich procházení. Druhá třída (`taskForm`) bude pracovat se zadanými daty, která ji budou předána pomocí formuláře z HTML stránky, a takto načtená data zpracuje a uloží do databáze. Obecně by se dalo říci, že třída `task` bude sloužit pro čtení z databáze a třída `taskForm` bude sloužit pro zápis dat do ní. Po delší úvaze nebyla nakonec na těchto třídách aplikována dědičnost, protože tyto třídy jsou natolik rozdílné, že by dědičnost nebylo vhodné použít. Ale jelikož třída `taskForm` bude obsluhovat formulář, tak bude potřebovat znát data z databáze, aby bylo možné určit, jestli se formulář změnil, zdali je správně vyplněn a jestli zadaný uživatel změnil jen položky, na které má právo. Proto pro správnou funkčnost třídy `taskForm` musíme předat ukazatel na vytvořený objekt z třídy `task`, aby mohla být čtena data z databáze a mohla být provedena kontrola formuláře.

4.2.2 Třída task

Tato třída bude určena pouze pro čtení dat o úkolech a stavech úkolů z databáze a nijak nebude měnit data, která se v databázi nacházejí. Protože je informační systém NLPIS velmi náročný co se počtu dotazů na databázi týče, tak se v této práci přikloníme k možnosti navrhnout tuto třídu tak, aby dokázala vybrat všechna potřebná data nejmenším možným počtem SQL dotazů, ideálně pouze jedním složitějším SQL dotazem. Takovýto dotaz na databázi sice zabere více času na zpracování, ale ve výsledku bude rychlejší než stovky menších dotazů, protože každý dotaz na databázi má ještě svoji režii, než se vůbec vykoná.

Pokud se změní jeden z úkolů v projektu, tak mají být vypsány vždy všechny úkoly, které jsou platné k aktuální změně úkolu. Tedy pokud projekt bude mít pět úkolů a jeden bude změněn, tak se má vypsat dvakrát pět úkolů, kde v jednom výpisu bude úkol před změnou a v druhém výpisu bude úkol po změně. Toto lze řešit buď v rámci třídy, nebo

později programově. Byla zvolena možnost, že toto bude řešit třída samotná a následně se to již nebude muset řešit programově při výpisu úkolů. Z tohoto důvodu bude potřeba načíst data z databáze do vnitřní proměnné a poté je vhodně seřadit, aby se dala jednoduše procházet dle potřeby. Návrh vnitřní datové struktury je vidět na obrázku č. 4.3.



Obrázek 4.3: Vnitřní datová struktura třídy task

Skupinou je myšlen hlavní celek, podle kterého se data budou vybírat. Bude se jednat o projekty a nebo o uživatele. Jednotlivé skupiny budou obsahovat 1-N změn. Počet těchto změn závisí na počtu rozdílných časů vložení úkolů pro danou skupinu. Kdykoliv se tedy vloží nový úkol s jiným časem než ostatní, tak bude vytvořena nová změna, ve které budou úkoly s platnými daty v době změny. Když budeme pokračovat dále v datové struktuře, tak dojdeme k jednotlivým úkolům, které budou platné v této změně. Každý z těchto úkolů bude mít stavy, které platí pro něj a platily i pro předchozí verze úkolů. Pokud tedy nějaký úkol má již několik stavů a my vytvoříme novou verzi tohoto úkolu, tak i nový úkol bude mít vypsané stavy z předešlé verze úkolu. Takto uložené stavy jsou nutné, protože někdy je potřeba vypsat pouze aktuální verzi úkolů a v tomto výpisu musí být uvedeny všechny stavy, kterými úkol i jeho předchůdci prošli.

Třída `task` bude obsahovat několik indexů, pomocí kterých se bude posouvat ve své vnitřní datové struktuře. Konkrétně každé zanoření bude mít svůj index, aby bylo možné se přehledně posouvat mezi daty. Po posunutí indexů na požadovaná data budou volány metody objektu, které budou vracet datové položky. Třída nebude vracet žádná HTML data, pouze hodnoty. Proto bude potřeba vytvořit funkce, které budou pracovat s třídou `task` a budou vkládat data do požadovaného HTML kódu. Díky tomuto bude možné oddělit

datovou část od vizuální části. Pro změnu vzhledu výpisu dat nebude potřeba měnit třídu samotnou, ale pouze funkci, která zajišťuje vypisování dat.

Informační systém NLPIS umožňuje uživateli filtrovat data a umožňuje jejich rozdělení na několik stran, dle pevného počtu zobrazených dat na stránce. Nabízejí se dvě možná řešení, jak tyto filtrace a řazení aplikovat na třídu `task`. Problém se bude řešit programově tak, že se přeskočí požadovaný počet řádků dat, která nás nezajímají, a poté se vypíšou pouze požadovaná data na stránku, nebo že třída `task` bude zohledňovat zadané filtrování a nastavené stránkování přímo v SQL dotazu, tedy vybere přesně požadovaná data. Jelikož se jedná o velké množství dat, která by SQL dotaz musel vracet, tak byla zvolena druhá možnost, díky které bude vykonaný dotaz rychlejší a efektivnější.

4.2.3 Třída taskForm

Hlavním úkolem třídy `taskForm` bude zpracování formuláře a aktualizace dat v databázi. Jak bylo zmíněno výše, pro správnou funkčnost kontroly formuláře bude potřeba předat při inicializaci objektu `taskForm` ukazatel na třídu `task`, aby mohla být čtena data z DB. Formulář bude nutno před odesláním zkontrolovat, jestli jsou zadané údaje správně vyplněny. Pokud údaje nebudou vyplněny správně, tak se tato skutečnost musí uživateli zobrazit ve formě výpisu chyb. Pro tento výpis bude potřeba udělat pole, ve kterém se budou uchovávat řetězce o nalezených chybách. Jelikož formulář musí podporovat přidávání a odebírání úkolů, bude potřeba využít JavaScript, aby uživatel kvůli přidání nebo odebrání úkolu nemusel obnovovat stránku.

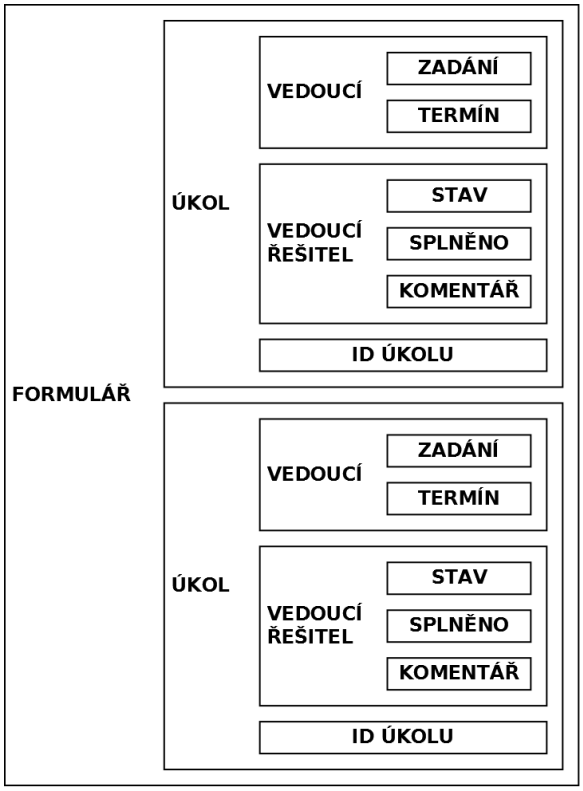
4.2.4 Formulář pro správu úkolů

Návrh formuláře je možné si prohlédnout na obrázku č. 4.4. Formulář se skládá ze dvou částí: jednu část může editovat pouze osoba s právy vedoucího a druhou část může editovat osoba jak s právy vedoucího, tak s právy řešitele. Každý úkol ve formuláři obsahuje šest položek.

Tyto položky obsahují:

- Zadání úkolu
- Termín úkolu
- Stav úkolu
- Procentuální splnění úkolu
- Komentář ke stavu úkolu
- Identifikátor úkolu v databázi

Identifikátor úkolu bude skrytý a bude obsahovat číselný identifikátor úkolu, který databáze přiřadila úkolu při jeho vytvoření. Tento identifikátor bude důležitý pro zpracování úkolu ve formuláři, protože všechny jeho hodnoty mohou být pozměněny a nemuseli bychom v databázi najít předchozí verzi úkolu. Při vložení nového úkolu, který se v databázi nenachází, bude hodnota tohoto identifikátoru nastavena tak, aby bylo třídě `taskForm` řečeno, že se jedná o nový úkol a má ho vložit a ne upravit.



Obrázek 4.4: Návrh formuláře

4.3 Přihlašování k úkolům na stránce MediaWiki

Na MediaWiki stránce jsou vypsány úkoly pro nové studenty a k těmto úkolům se studenti přihlašují tak, že za název úkolu vloží svůj login. Poté je úkolem vedoucího, který úkol zadal, aby si hlídal, jestli se mu na některý z úkolů nepřihlásil nějaký student. Tento proces je velmi zdlouhavý a jedním z naších úkolů zautomatizovat, aby po přihlášení řešitele byl vedoucí zadaného úkolu upozorněn, že u jeho úkolu existuje nový řešitel.

Jelikož lze měnit data wiki úkolů jak v systému MediaWiki, tak v informačním systému NLPIS, tak bude zapotřebí určit správný zdroj dat, aby nedošlo k možné ztrátě informací, pokud se budou lišit data v databázi a na MediaWiki stránce. K tomuto účelu bude v databázi uložen atribut, který bude informovat o tom, v jakém stavu se wiki úkol nachází. Dle tohoto stavu bude určeno, který ze zdrojů bude mít při zpracování dat přednost. Ale i přes toto řešení může dojít k situaci, kdy nebude jasné, který ze zdrojů dat je aktuální. Touto situací je stav, kdy jsou zároveň změněna data jak v databázi, tak na stránce MediaWiki. V tomto případě je určeno, že přednost má vždy stránka MediaWiki. V takové situaci tedy bude databáze přepsána daty, která se nachází na MediaWiki stránce.

4.3.1 Stránka pro přihlašování k úkolům

Pro uchování informací o přihlášených řešitelích a obsahu MediaWiki stránky pro přihlašování k úkolům bude zapotřebí vytvořit databázové tabulky. Díky těmto tabulkám bude možné upravovat wiki úkoly a jejich řešitele přímo v informačním systému NLPIS. Tyto tabulky jsou zobrazeny na obrázku č. 4.5. Tabulka `wiki_ukoly_sekce` uchovává struk-

turu MediaWiki stránky. Jsou v ní uloženy sekce a skupiny na které je MediaWiki stránka rozdělena. Tabulka `wiki_ukoly` bude uchovávat údaje o jednotlivých úkolech, které jsou na stránce vloženy. V tabulce `wiki_registrace` pak budou uloženi registrovaní řešitelé k jednotlivým úkolům.

Obrázek 4.5: Databázové tabulky vytvořené pro přihlašování k wiki úkolům

4.3.2 Třídy pro přihlašování k úkolům na stránce MediaWiki

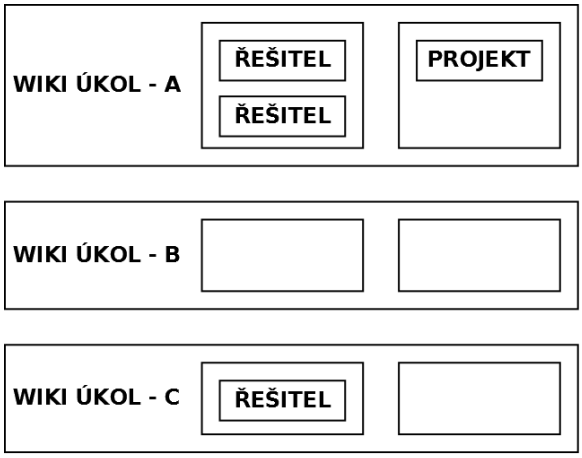
Kód, který bude obsluhovat úkoly pro nové studenty, bude objektově orientovaný a obdobně jako u správy úkolů v systému NLPIS budou dvě třídy, kdy opět jedna z tříd bude sloužit pouze ke čtení dat z databáze a jejich následnému setřídění dle potřeby (třída `taskWiki`) a druhá třída bude sloužit pro kontrolu a zpracování obsahu stránky MediaWiki (třída `taskNLP`). Tyto třídy jsou zobrazeny na obrázcích v příloze B. Z pohledu databáze by se dalo říct, že třída `taskWiki` slouží pouze pro čtení dat z databáze a třída `taskNLP` slouží k zápisu dat do databáze. Tyto třídy jsou na sobě navzájem nezávislé a mohou fungovat samostatně, tedy není potřeba předávat žádné ukazatele na vytvořené objekty.

4.3.3 Třída taskWiki

Hlavním účelem třídy `taskWiki` bude čtení dat o úkolech pro nové studenty a jejích řešitelích z databáze. Třída nijak nemodifikuje data, která se v databázi nacházejí. Jelikož informační systém NLPIS obsahuje velký počet dotazů na databázi, tak v této třídě bude výběr dat pouze pomocí jednoto SQL dotazu. Tento dotaz sice může vybrat redundantní data, ale ve výsledku bude tento SQL dotaz proveden rychleji než větší počet menších dotazů na databázi.

Provedený SQL dotaz bude vybírat data, která se musí nejprve projít a roztřídit dle jednotlivých úkolů. Toto roztřídění se musí provádět, protože výsledný SQL dotaz bude vracet redundantní data, která vznikla z důvodu, že jeden wiki úkol může obsahovat více řešitelů a zároveň z jednoho wiki úkolu mohlo vzniknout několik projektů. Tedy každý wiki úkol se bude ve výsledku dotazu vyskytovat dle součinu $(M \cdot N)$, kde M je počet řešitelů wiki úkolu a N je počet projektů, které z wiki úkolu vznikly, a kde $M > 0$ a $N > 0$. Pokud jedna z položek je nulová, tak se wiki úkol vyskytne ve výsledku M-krát, respektive N-krát. Pokud jsou nulové obě položky, tak se ve výsledku dotazu zobrazí wiki úkol pouze jednou.

Takto roztříděná data budou uložena do vnitřní struktury třídy `taskWiki`, která je zobrazena na obrázku č. 4.6. Zde lze vidět, že každý wiki úkol bude mít dvě pole, ve kterých jsou uloženy informace o jednotlivých přihlášených řešitelích a informace o projektech, které vznikly z tohoto wiki úkolu. Pro pohodlnější práci s načtenými daty se bude uživatel posouvat pomocí indexů, které budou ukazovat na požadovaná data. Indexy budou konkrétně tři, z toho index pro řešitele a index pro vzniklé projekty budou na sobě nezávislé (protože nemusí být nastaven jeden index pro čtení dat z druhého indexu). Obdobně jako u třídy `task` i tato třída nebude vracet žádná HTML data, pouze hodnoty načtené z databáze. Aby mohla být v informačním systému NLPIS využita tato třída, je potřeba vytvořit funkce, které budou s touto třídou pracovat a budou vracet požadovaný HTML kód. Díky této koncepci bude oddělená datová část od vizuální části a pro změnu HTML kódu a pro výpis dat nebude třeba měnit kód třídy, ale pouze kód dané funkce, která obsluhuje vypsání dat.



Obrázek 4.6: Vnitřní datová struktura třídy taskWiki

Obdobně jako u třídy `task` i tato třída musí umožňovat příjem nastavených filtrů a stránkování, které si uživatel může nastavit v informačním systému NLPIS. Třída `taskWiki` tyto nastavené filtry zakomponuje do svého SQL dotazu tak, aby databáze vracela opravdu pouze data, která jsou potřeba, a byla tak maximalizována rychlost a efektivnost SQL dotazu.

4.3.4 Třída taskNLP

Tato třída bude určena pro zpracování MediaWiki stránky, kde se přihlašují studenti na vypsané úkoly pracovní skupiny NLP a na aktualizaci dat v databázi. Konkrétní MediaWiki stránka, která se bude zpracovávat, má pevně danou strukturu, která je zobrazena na obrázku číslo 2.1.

Pro procházení MediaWiki stránky bude zapotřebí využít konečného automatu, díky kterému budeme moci zároveň kontrolovat strukturu stránky a provádět operace nad databází. Návrh konečného automatu je zobrazen na obrázku č. 4.7. Pro snadnější čtení obsahu stránky MediaWiki bude načtený řetězec rozdělen dle znaku konce řádku na jednotlivé řádky. Díky tomu bude konečný automat snadněji procházet načtený text, protože bude pro určení stavu stačit načítat začátek řetězce na řádku a dle toho se určí, do jakého stavu se má přejít.

4.3.5 Konečný automat třídy taskNLP

Výchozím stavem konečného automatu (dále jen KA) je stav `unknown`. V tomto stavu se KA rozhoduje, ke kterému stavu náleží řetězec, který se nachází na začátku aktuálně zpracovávaného řádku. Ze stavu `unknown` může konečný automat přejít do devíti dalších očekávaných stavů. Pokud se neobjeví žádný z těchto stavů, tak KA přejde do chybového stavu nazvaného error. V tomto chybovém stavu se načtou další řádky souboru, dokud se nenarazí na další celek, který se nachází na úrovni chybně načteného celku. Tedy pokud vznikne chyba při čtení skupiny, tak se bude číst tak dlouho, dokud se nenarazí na další skupinu nebo na konec souboru. Řetězec, jehož zpracování se takto přeskočí, bude přesunut do chybné sekce, která je umístěna na konci každé skupiny nebo sekce.

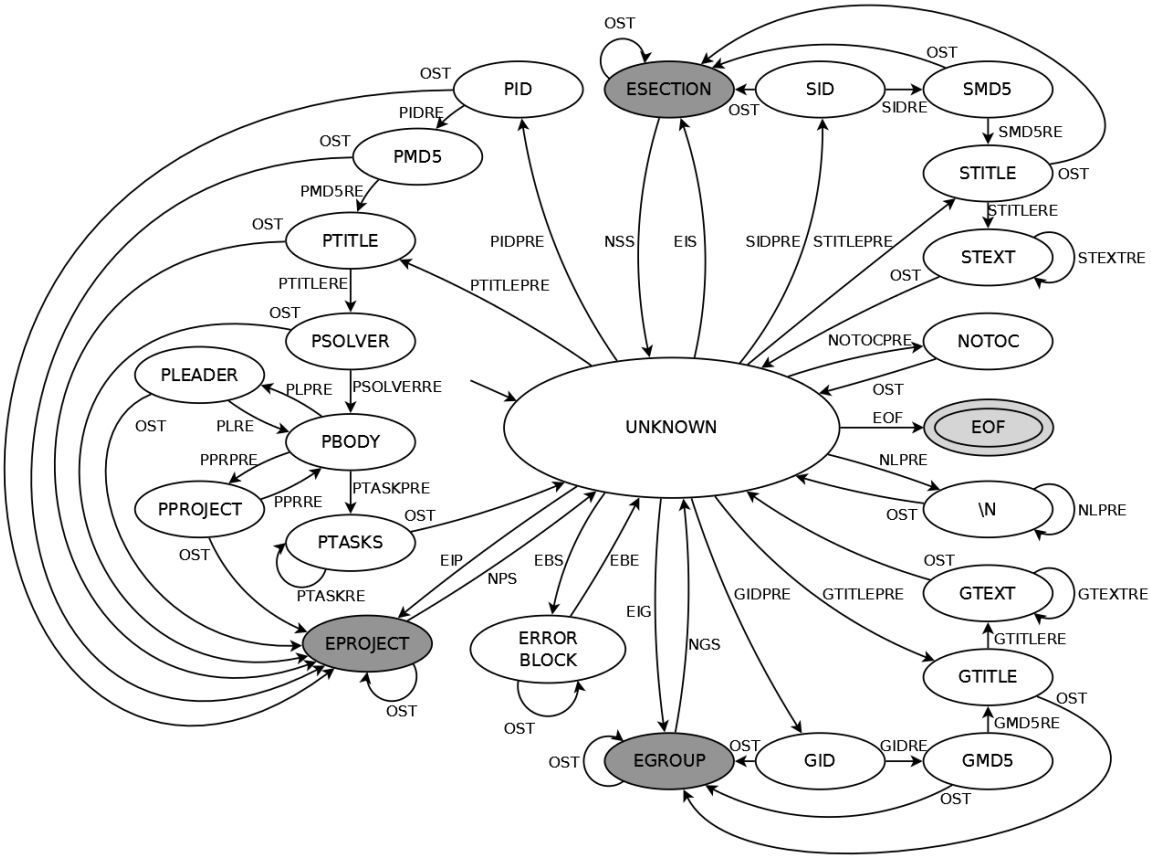
Sekce může na stránce začínat buď specifickým nadpisem pro sekci, nebo HTML komentářem, který v sobě obsahuje ID sekce. Tento komentář bude vložen při generování stránky a díky tomuto komentáři se pozná, jestli se jedná o novou sekci, nebo o sekci, která již v databázi existuje, a dle toho se také bude k dané sekci chovat (bude vkládat nebo editovat). Aktuálně se na stránce pro nové úkoly nachází pouze jedna sekce, ale je bráno v úvahu možné budoucí rozšíření a proto bude třída `taskNLP` připravena, aby byla schopna obsluhovat více sekcí na stránce. Pokud sekce bude obsahovat identifikátor, tak také bude obsahovat MD5 hash, který byl vygenerován při generování stránky. Když budeme znát ID sekce a budeme mít hash z textu, který byl uložen při generování, tak se z databáze načtou údaje o této sekci, vygeneruje se text z aktuálních dat v databázi, který odpovídá textu vkládanému do této MediaWiki stránky, a následně se z tohoto vygenerovaného textu spočítá MD5 hash. MD5 hash se zároveň vypočítá z aktuálního obsahu MediaWiki stránky, který se vztahuje k této sekci. Celkově tedy budou tři MD5 hashe. První hash je pro aktuální verzi textu na MediaWiki stránce, druhý hash je z textu, který vznikl z aktuálních dat z databáze, a třetí hash je uložen na MediaWiki stránce a představuje hash, který odpovídal (a stále může odpovídat) textu, který byl v MediaWiki uložen při generování stránky. Porovnáním těchto hashů zjistíme, jestli došlo k nějaké změně dat na stránce MediaWiki nebo ke změně dat v databázi. Díky tomuto víme, který ze zdrojů dat obsahuje aktuálnější verzi a jestli se má případně přepsat obsah databáze nebo obsah stránky v MediaWiki. Když sekce identifikátor neobsahuje, tak neobsahuje ani vložený MD5 hash a z toho tedy vyplývá, že se jedná o nově vloženou sekci, která má být následně vložena do databáze a její přiřazený identifikátor v databázi a vypočtený MD5 hash mají být vloženy do MediaWiki stránky. Pokud je identifikátor, hash nebo nadpis sekce špatně vyplněn, tak dojde k přechodu do chybné sekce a celá sekce bude označena za chybnou.

Obdobná situace nastává u skupiny. Skupina také začíná nadpisem, nebo identifikátorem, který je následován MD5 hashem. Po nadpisu skupiny následuje volitelný popis skupiny. Pro zjištění, zda nastala změna dat v databázi nebo na MediaWiki stránce, bude sloužiz stejný postup jako u sekce, tedy porovnáním tří MD5 hashů. Pokud bude identifikátor, hash nebo nadpis ve špatném formátu, tak se stav KA změní na chybový stav a opět bude celá skupina přesunuta do chybové sekce, která je umístěna na konci sekce.

Zpracování projektu na MediaWiki stránce bude komplikovanější, protože projekt obsahuje nejvíce informací a z toho některé nemusí být vyplněny. Počátek celku označující projekt začíná buď speciálním HTML komentářem, který obsahuje identifikátor projektu, nebo nadpisem projektu v případě, když je na MediaWiki stránku vložený nový projekt. Detailnější informace o struktuře projektu na stránce MediaWiki jsou vypsány v kapitole 4.3.1.

Dalšími stavy, které bude KA obsahovat, jsou stavy `NOTOC`, čtení prázdného řádku a čtení

chybné sekce. Ve stavu `NOTOC` se čte speciální řetězec MediaWiki, který odstraňuje na začátku stránky tabulku obsahu. Stav čtení prázdného řádku je potřebný, aby nedocházelo k falešnému vyhodnocení chybného stavu. Jelikož chybná sekce obsahuje části stránky, které se nepodařilo přečíst, tak tuto sekci při čtení stránky vynecháme a pokračujeme v dalším čtení stránky až po ukončení této sekce. Vynechání chybné sekce je nutné, protože by při každém čtení stránky byly znovu nalezeny chyby přesunuté do této sekce.



Obrázek 4.7: Konečný automat pro zpracování MediaWiki stránky s úkoly pro nové řešitele

4.3.6 Popis konečného automatu

SIDPRE	– řetězec na začátku řádku označuje identifikátor sekce
STITLEPRE	– řetězec na začátku řádku označuje nadpis sekce
GIDPRE	– řetězec na začátku řádku označuje identifikátor skupiny
GTITLEPRE	– řetězec na začátku řádku označuje nadpis skupiny
PIDPRE	– řetězec na začátku řádku označuje identifikátor projektu
PTITLEPRE	– řetězec na začátku řádku označuje nadpis projektu
NOTOCPRE	– řetězec na začátku řádku označuje řetězec NOTOC
NLPRE	– je načten prázdný řádek
EOF	– je načten konec souboru
EIS	– nastala chyba v sekci

EIG – nastala chyba ve skupině
EIP – nastala chyba v projektu
EBS – řetězec na začátku řádku označuje začátek bloku s chybami
SIDRE – řetězec vyhovuje identifikátoru sekce
SMD5RE – řetězec vyhovuje MD5 hashi sekce
STITLERE – řetězec vyhovuje nadpisu sekce
STEXTRE – řetězec vyhovuje textovému popisu sekce
GIDRE – řetězec vyhovuje identifikátoru skupiny
GMD5RE – řetězec vyhovuje MD5 hashi skupiny
GTITLERE – řetězec vyhovuje nadpisu skupiny
GTEXTRE – řetězec vyhovuje textovému popisu skupiny
PIDRE – řetězec vyhovuje identifikátoru projektu
PMD5RE – řetězec vyhovuje MD5 hashi projektu
PTITLERE – řetězec vyhovuje nadpisu projektu
PSOLVERRE – řetězec vyhovuje řešiteli projektu
PLPRE – řetězec na začátku řádku označuje vedoucího projektu
PLRE – řetězec vyhovuje vedoucímu projektu
PPRPRE – řetězec na začátku řádku označuje skupinu projektu
PPRRE – řetězec vyhovuje skupině projektu
PTASKPRE – řetězec na začátku řádku označuje zadání úkolu projektu
PTASKRE – řetězec vyhovuje zadání úkolu projektu
NSS – nalezen začátek nové sekce nebo EOF
NGS – nalezen začátek nové skupiny, sekce nebo EOF
NPS – nalezen začátek nového projektu, skupiny, sekce nebo EOF
EBE – řetězec na začátku řádku označuje konec bloku s chybami
OST – řetězec nevyhovuje ostatním přechodům v aktuálním stavu

4.3.7 Gramatika stránky pro přihlašování k úkolům

<stranka> → <sekce>
<sekce> → <sekce_nadpis><sekce_popis><skupina><sekce>
<sekce> → <sekce_nadpis><sekce_popis><skupina>
<sekce> → <sekce_nadpis><sekce_popis>
<skupina> → <skupina_nadpis><skupina_popis><projekt><skupina>
<skupina> → <skupina_nadpis><skupina_popis><projekt>
<skupina> → <skupina_nadpis><skupina_popis>
<projekt> → <projekt_nadpis><projekt_resitele><projekt_vedouci>
<projekt_projekt><projekt_ukol><projekt>
<projekt> → <projekt_nadpis><projekt_resitele><projekt_vedouci>
<projekt_projekt><projekt_ukol>
<sekce_nadpis> → ==<text>==
<sekce_popis> → <text>
<skupina_nadpis> → ===<text>===
<skupina_popis> → <text>
<projekt_nadpis> → "'<text>"'
<projekt_resitele> → (<login>)
<login> → <text><login2>

<login> → <text>
<login2> → , <text>
<projekt_vedouci> → vedoucí: <vedouci>
<projekt_vedouci> → vedoucí: <vedouci>, <vedouci>
<vedouci> → <text> (<email>)
<projekt_projekt> → projekt: <text>
<projekt_ukol> → *<text>

Kapitola 5

Implementace

Implementace v této bakalářské práci znamenala vytvořit vlastní třídy, které jsou schopny spravovat úkoly v systému NLPIS, dále upravit funkcionalitu diplomové práce pána Ing. Aleše Vavřínka a následně tuto práci ještě rozšířit o nově vytvořené třídy. V neposlední řadě bylo nutné upravit stávající zdrojové soubory systému NLPIS, do kterých jsem přidal již zmíněné třídy pro správu úkolů, a následně rozšířit informační systém NLPIS o nové stránky, kde jsou informace z tabulek, které byly přidány v rámci bakalářské práce. Pro snadnější a přehlednější integraci do stránek systému NLPIS, byl vytvořen soubor funkcí, které vrací HTML kód, který se vloží do požadovaného místa na stránce. Díky tomu není potřeba rozsáhlých změn v kódu informačního systému a navíc jsou všechny funkce, které obsluhují mé třídy, v rámci jednoho souboru, což přináší značené usnadnění práce při editaci vzhledu systému NLPIS.

5.1 Automatické generování wiki stránek z e-mailů a IS

Implementace změn týkajících se této části byla velmi zdlouhavá a zabrala více času, než bylo předpokládáno. Funkce, které obsluhovaly kontrolu a generování projektů a skupin se jmenují `kontrola_stranky_projektu`, `vytvoreni_stranky_projektu` a `kontrola_stranky_skupiny`. Funkce pro vytvoření stránky skupiny a projektu byly ponechány beze změn, protože tyto funkce jsou volány pouze pokud stránka s projektem nebo se skupinou neexistuje.

5.2 Správa úkolů v systému NLPIS

Pro správu úkolů v systému NLPIS byly vytvořeny dvě třídy. První třída dostala název `task`, protože čte úkoly z databáze. Druhá třída dostala název `taskForm`, jelikož zpracovává formulář s úkoly a dle potřeby aktualizuje databázi.

5.2.1 Třída task

Třída `task` implementuje čtení dat z databáze. Není tedy nutné pro čtení dat o úkolech vytvářet SQL dotazy na databázi. Tato třída je zkonstruována tak, aby poskytla veškerá data, která jsou potřebná pro výpis informací o úkolech, jejich řešitelích a ostatních informacích, které jsou podstatné. Jelikož metoda `getTasks` je stavěna jako víceúčelová, aby ji bylo možné využít pro všechny potřebné pohledy na data, byla vytvořena sadu metod, které obsahují jen potřebné parametry k danému pohledu na data.

Tyto metody jsou:

- `getAllTasks` - Z databáze jsou vybrány všechny úkoly a k ním příslušející projekty a řešitelé. Tato metoda je využita na stránce s úkoly, kde jsou vypsány všechny úkoly, které mohou být dále omezeny zadaným uživatelským filtrem.
- `getTasksByID` - Vybere konkrétní úkol dle jeho identifikátoru, který se předá jako parametr. V informačním systému NLPIS tato metoda není využita. Byla přidána z důvodu možného budoucího využití. Metoda by se měla volat co nejméně, pokud chceme načíst data úkolů celého projektu, tak využijeme jinou metodu a nebudeme několikrát volat tuto, protože její SQL dotaz na databázi je přibližně stejně komplikovaný jako dotaz pro zjištění úkolů celého projektu.
- `getTasksByUser` - Parametrem této metody je identifikátor uživatele, pro kterého chceme nalézt všechny jeho řešené projekty. Tato metoda se hodí, pokud uživatel chce vidět všechny své řešené projekty.
- `getTasksByProject` - Tato metoda vybere všechny řešitele pro projekt, který je specifikovaný parametrem. Samozřejmě jsou zároveň vybrány informace o projektu a o úkolech jednotlivých řešitelů.
- `getTasksByUserProject` - Spojením dvou předchozích metod vznikne uvedená metoda, která vybere data pro konkrétní úkol a konkrétního řešitele. Tato metoda je nejvíce využívána pro práci s formulářem, který obsluhuje úkoly.

Každá z těchto metod pak dále může obsahovat nepovinné parametry, které jsou využity pro filtrování a řazení nalezených dat. Tyto výše zmíněné metody pouze volají hlavní metodu `getTasks`, které předají potřebné parametry a tato metoda se již postará o naplnění vnitřní datové struktury daty z SQL dotazu. Metody pro výběr dat vrací pravdivostní hodnotu, kterou říkají, jestli se data podařilo vybrat.

5.2.2 Čtení dat z databáze

Ze začátku se inicializují vnitřní proměnné objektu `task` a vyprázdní se vnitřní datová struktura. Poté se dle předaných parametrů vyhodnotí, o jaký pohled na data se jedná, a následně se upraví SQL dotaz, aby odpovídal zvolenému náhledu. Následuje volání SQL dotazu pro výběr potřebných dat. Tento SQL dotaz je velmi rozsáhlý a obsahuje několik poddotazů. Tyto poddotazy musely být přidány z důvodu stránkování. Stránkování se vždy dělá na určitý počet prvků na stránce, ale nově udělaný SQL dotaz nevybírá pouze jednotlivé prvky (projekty, řešitele), ale i jejich úkoly. Tedy nelze jednoduše omezit SQL dotaz na 50 záznamů, protože 50 záznamů odpovídá 50ti úkolům a to může být například 7,5 projektu. Tedy by na stránce bylo vypsáno 8 projektů (nebo řešitelů) a u posledního by se vypsaly jen některé úkoly. Navíc by takovéto stránkování fungovalo špatně. Proto se poddotazem musí vyfiltrovat požadovaný počet řešených projektů a k ním přiřazených řešitelů (tabulka `resi`). Ale protože se řazení provede až po provedení klauzule Group By, tak je potřeba vybrat všechny identifikátory projektů a identifikátory řešitelů a v dalším poddotazu je pomocí klauzule Limit omezit na požadovaný počet prvků. Při implementaci byl nalezen problém s MySQL databází, kdy se zadaný SQL dotaz nemohl provést, protože MySQL nepodporuje klauzuli Limit v poddotazu. Řešením tohoto problému bylo zakomponovat ještě jeden poddotaz, který bude zastřešovat poddotaz s klauzulí Limit a pouze předá identifikátory

projektu a identifikátory řešitelů hlavní části SQL dotazu, která již vybere všechny potřebné atributy. Celkem tento SQL dotaz obsahuje tři vnořené poddotazy.

Po vykonání SQL dotazu je potřeba projít všechna načtená data. Proto v cyklu procházíme načtené záznamy z databáze a rozdělujeme je do skupin dle identifikátoru projektu a identifikátoru řešitele, čímž je docílíme toho, že dostaneme úkoly, které patří k jednomu projektu a řešiteli, do jednoho pole. Zároveň se ukládají všechny rozdílné časové údaje pro jednotlivé skupiny. Tyto časové údaje budou poté využity pro tvorbu pole jednotlivých změn. Když již máme rozdělena data dle skupin a známe všechny časové změny, které nastaly v jednotlivých skupinách, tak můžeme začít plnit vnitřní datovou strukturu. Naplňování této datové struktury má formu několika vnořených cyklů. První cyklus prochází jednotlivé skupiny. Druhý cyklus prochází časové změny, které nastaly ve skupině. Jelikož by se mohlo stát, že úkoly nejsou z databáze vráceny přesně v požadovaném pořadí, tak je zde vložen další cyklus, který procházejí cyklicky pole a hledá postupně (dle atributu `poradi`) úkoly, které náleží dané skupině (projektu a řešiteli). Pokud by nastala nekonzistence databáze, která by se projevila špatně uloženým pořadím úkolů, tak se vypíšou pouze úkoly, které mají správně definované pořadí. Pro lepší vysvětlení si uvedeme příklad. Pokud budou v databázi existovat úkoly s pořadím 1, 2, 3 a 5, tak budou vypsány pouze úkoly s pořadím 1, 2 a 3. V tomto cyklu se taky pro každou změnu načítají stavy jednotlivých úkolů. Toto je ale náročnější, co se počtu dotazů na databázi týče, a proto se provádí pouze pokud si uživatel přeje zobrazovat historii stavů. Pokud si uživatel přeje zobrazit pouze poslední (aktuální) stav úkolu, tak ten se načítá z databáze již v dříve zmiňovaném SQL dotazu. Díky možnosti zákazu výběru historie stavů lze částečně zkrátit dobu vykonávání SQL dotazu. Díky těmto vnořeným cyklům vznikne datová struktura, která je zobrazena na obrázku č. 4.3.

5.2.3 Počet prvků v databázi

Pro správnou funkci stránkovače na stránkách informačního systému NLPIS je potřeba znát celkový počet záznamů. Kvůli tomuto obsahuje třída `task` podobné metody jako pro výběr dat z databáze. Tyto metody mají názvy `countAllTasks`, `countTasksByUserProject`, `countTasksByUser`, `countTasksByProject` a `countTasksByID` a odpovídají výše uvedeným metodám, které vybírají data. Jejich parametry jsou také shodné. Jediným rozdílem je, že tyto metody nevrací pravdivostní hodnotu, jestli se podařilo data vybrat, ale počet nalezených záznamů. Podobně jako u metod pro výběr dat, i tyto metody volají pouze metodu `countTasks`, které předají potřebné parametry, včetně pohledu na data, který se má použít.

Metoda `countTasks` má vrátit počet nalezených záznamů v databázi. Postup, aby toho docílila, je následující: Nejprve se zpracují předané parametry, dle kterých se upraví následný SQL dotaz, aby odpovídal danému pohledu na data a bylo aplikováno případné filtrování dat. Řazení dat se v tomto dotazu neřeší, protože nás zajímá počet, který není ovlivněn směrem řazení dat. Je ale možné využít stránkování, přičemž v takovém případě se metoda nedá použít pro zjištění celkového počtu prvků v databázi, ale k celkovému počtu prvků, který se bude nacházet na stránce. Mohlo by to sloužit k určení počtu prvků na poslední stránce ještě před samotným výběrem dat. Takto upravený SQL dotaz bude zpracován a bude zjištěn počet záznamů, které byly v databázi nalezeny. Popis SQL dotazu je shodný s popisem SQL dotazu v kapitole 5.2.2. Jedinou odlišností mezi těmito SQL dotazy je výběr dat. Pro efektivnější a rychlejší zpracování tento SQL dotaz nevybírá data (je vybrána konstanta o hodnotě 1).

5.2.4 Metody třídy task

Třída `task` obsahuje různorodý výčet metod pro posun indexů v polích a tedy různé možnosti, jak procházet načtenými daty. Tyto metody jsou potřebné z důvodu uspořádání vnitřní datové struktury, která je složena z několika zanořených polí. Indexy existují v objektu `task` čtyři a mají názvy `groupIndex`, `changeIndex`, `taskIndex` a `stateIndex`.

Každý z těchto indexů má tyto typy metod pro změnu jeho hodnoty:

- nastaví index na hodnotu -1 (`resetGroup`, `resetChange`, `resetTask`, `resetState`)
- inkrementuje index (`nextGroup`, `nextChange`, `nextTask`, `nextState`)
- dekrementuje index (`prevGroup`, `prevChange`, `prevTask`, `prevState`)
- nastaví index na hodnotu 0 (`firstGroup`, `firstChange`, `firstTask`, `firstState`)
- nastaví index na poslední prvek (`lastGroup`, `lastChange`, `lastTask`, `lastState`)
- nastaví index na zvolený prvek (`setGroup`, `setChange`, `setTask`, `setState`)

Tyto metody vrací pravdivostní hodnotu, která označuje, jestli se povedlo změnit index (jestli požadovaný index existuje). Výchozí hodnota indexu je -1, aby tyto metody mohly být využity v programové konstrukci `while` místo konstrukce `do-while`. V bakalářské práci jsou využity pouze metody typu next, first a reset, ostatní metody jsou dodány pro případné budoucí rozšíření.

Třída `task` také umožňuje vyhledávání v datech načtených z databáze. Je důležité mít na paměti, že použití vyhledávací metody funguje tak, že po zavolání nastaví index skupiny nebo úkolu na hledaný prvek. Tedy dojde k přepsání aktuálních indexů v objektu.

K hledání existují 3 metody:

- `findProject` - Parametrem je identifikátor projektu, který se má nalézt. Pokud metoda nalezne hledaný projekt, tak vrátí pravdivostní hodnotu true a nastaví pouze `groupIndex` na nalezenou pozici. Pro správné čtení dat je třeba nastavit `changeIndex`, `taskIndex` a případně `stateIndex`.
- `findSolver` - Metoda jako parametr očekává identifikátor řešitele, který se má najít. Pokud bude hledaný řešitel nalezen, tak bude navrácena pravdivostní hodnota true a index `groupIndex` bude nastaven na pozici nalezeného řešitele. Pro čtení dat je třeba nastavit `changeIndex`, `taskIndex` a případně `stateIndex`.
- `findTask` - Parametrem je identifikátor úkolu, který chceme najít v aktuální skupině a v aktuální změně. Tato metoda pouze pohybuje indexem `taskIndex`. Pro případné čtení stavů úkolů je potřeba nejprve nastavit `stateIndex`.

Všechny tyto metody mají jako nepovinný parametr start, který označuje počáteční index, od kterého se má začít hledání. Toto lze využit pro postupné hledání, kdy chceme nalézt jiný než první prvek (když se vyberou všechny úkoly, tak jeden projekt má více řešitelů a jeden řešitel má více projektů). Tento počáteční index může být zadán jako kladné číslo, kdy tento index bude znamenat počáteční pozici od začátku pole, nebo jako záporné číslo, kdy tento index bude znamenat počáteční pozici od konce pole (hodnota -2 znamená, že se začíná hledat od předposledního prvku).

Výběr dat se provádí pomocí get metod, kdy každá proměnná má svojí vlastní metodu. Pokud nejsou řádně nastavené potřebné indexy pro čtení dat, tak bude vytvořena

výjimka, konkrétně `OutOfRangeException`. Kdyby z nějakého důvodu tento způsob získávání dat nevyhovoval, tak je možné využít metodu `getTaskData`, která vrátí pole, které obsahuje data k aktuálně zvolenému úkolu, nebo metodu `getStateData`, která vrací pole se stavy aktuálně zvoleného úkolu. Těmto metodám taky lze dle potřeby nastavit indexy, ze kterých se má číst požadovaný úkol nebo stav. Pokud kombinace zadaných indexů neexistuje, tak bude vytvořena výjimka, konkrétně výjimka `OutOfRangeException`. Index může být zadán kladnou hodnotou, takže bude reprezentovat číslo indexu, které se má zvolit. Dále může být index reprezentován i záporným číslem, kdy udává index počítaný od konce (hodnota -2 představuje předposlední prvek).

Další metody, které stojí za zmínku, jsou metody pro překlad a výpis celého jména. Metody pro překlad jsou využívány pro překlad číselné hodnoty na příslušný textový řetězec. Jména těchto metod jsou `translateTaskState` a `translateProjectState`. Tyto metody vrací přeložený číselný stav aktuálního úkolu, respektive projektu. Díky volitelnému parametru lze přeložit zadané číslo na příslušný textový stav. Pokud nebude možné provést překlad na textový řetězec, tak bude vytvořena výjimka. Metody pro výpis celého jména jsou užitečné hlavně proto, že místo volání čtyř metod, stačí volat pouze jednu. Tyto metody jsou tři a mají názvy `getProjectWholeUserName`, `getTaskWholeSolverName` a `getStateWholeAuthName`. Všechny tyto tři metody mají volitelný textový parametr, který je využit jako oddělovač mezi jednotlivými částmi jména. Pokud není oddělovač nastaven, tak se použije znak mezery.

5.2.5 Třída taskForm

Třída `taskForm` implementuje zpracování formuláře pro správu úkolů a zápis dat do databáze. Pro správnou funkci této třídy je zapotřebí mít vytvořený objekt třídy `task`, který se předává do konstruktoru. Dalším povinným parametrem konstruktoru je číselný identifikátor osoby, která vyžaduje vykreslení formuláře (návštěvník stránky). Tento identifikátor je později použit při vyhodnocení, jestli daný uživatel má právo měnit úkoly a stavy úkolů. Další parametry jsou nepovinné a lze s nimi měnit názvy položek, se kterými má objekt `task` pracovat. První volitelný parametr udává název tlačítka (název indexu v globální proměnné `$_POST`), které udává, jestli byl formulář odeslán. Hodnota může být zadána jako řetězec, který označuje jeden konkrétní input, nebo jako pole řetězců, které označuje více inputů a pokud bude jeden z těchto inputů naplněn daty, tak se formulář vyhodnotí jako odeslán. Pokud bylo potřeba zkontrolovat, jestli určitý input ze zadaného pole byl odeslán (naplněn), tak lze využít metodu `isSend`, které se předá číselný parametr, který určuje index v poli a definuje, který input se má zkontrolovat.

5.2.6 Kontrola a zpracování formuláře

Před zpracováním formuláře je potřeba formulář zkontrolovat, jestli byly provedeny pouze změny, na které má uživatel právo, a jestli je odeslaný formulář nepoškozený. Poškozeným formulářem rozumíme formulář, jehož položky byly pozměněny pomocí upraveného HTML kódu, jako to umí například doplněk do prohlížeče FireFox zvaný FireBug. Metoda `checkForm` kontroluje, jestli prohlížející uživatel mohl provádět změny ve formuláři a jestli to, co neměl právo změnit, zůstalo stejné jako před odesláním. Dále se kontroluje správnost zadaných dat jako například rozsahy čísel, jestli text neobsahuje nedovolené znaky apod.

Kontrola na podvržení formuláře se provádí při ukládání úkolů v metodě `saveTasks`. Kontrolovány jsou identifikátory úkolů a pokud se zjistí, že uživatel se pokouší ve formuláři

upravit úkol, který nepatří k danému projektu, tak formulář nebude uložen a bude nahlášena chyba o integritě formuláře. Tento problém může nastat ve dvou případech. První možnost je, že uživatel záměrně pozměnil formulář a druhá možnost je, že někdo v databázi přepsal identifikátor úkolu nebo úkol z databáze smazal mezitím, co uživatel měnil formulář. Protože se může vyskytnout problém během zpracovávání formuláře, tak jsou všechny SQL dotazy ukládány do pole a pokud nebude nalezena žádná chyba, tak budou postupně tyto SQL dotazy provedeny.

5.2.7 Chybová hlášení

Uživatel ne vždy vyplní data správně a důsledkem toho je, že formulář nelze odeslat. Pro tyto případy má třída `taskForm` pole, které obsahuje případné chybové řetězce. Pro zjištění, jestli nějaké chyby existují, slouží metoda `hasError` a pro přečtení chybové zprávy se využívá metoda `readError`. Jelikož jsou chybové řetězce uloženy v poli, tak je potřeba v tomto poli posunovat index. K tomuto účelu jsou využity obdobné metody jako u třídy `task`.

Jedná se o metody:

- `nextError` - Posune index na další chybu a vrátí pravdivostní hodnotu, jestli se index podařilo posunout. Pokud tedy již žádná další chyba neexistuje, bude vrácena hodnota false.
- `prevError` - Přesune aktuální index na předchozí chybu a vrátí hodnotu true. Pokud nelze index posunout, bude navrácena hodnota false.
- `firstError` - Pokud existuje chyba, tak bude index posunut na první chybu v poli a bude navrácena hodnota true, jinak bude pouze navrácena hodnota false.
- `lastError` - Index bude nastaven na poslední chybu v poli. Když je pole prázdné, tak bude navrácena hodnota false.
- `resetError` - Nastaví index na hodnotu -1, aby při dalším volání metody `nextError` mohla být přečtena první chyba.

Za všechny nastalé chyby nemusí být odpovědný uživatel. Mohou také nastat chyby při provádění SQL dotazů. Pro tento případ má třída `taskForm` další pole, které obsahuje provedené SQL dotazy a posledním vloženým SQL dotazem je právě onen SQL dotaz, který vyvolal MySQL chybu. Jelikož MySQL na cílovém serveru nepodporuje transakce, tak pomocí těchto dotazů může administrátor vrátit provedené změny (pokud jsou známy předchozí hodnoty). V poli chybných SQL dotazů se posouváme pomocí metod `nextSQLError`, `prevSQLError`, `firstSQLError`, `lastSQLError` a `resetSQLError`. Funkce těchto metod jsou obdobné jako metody pro chybová hlášení.

Třída `taskForm` obsahuje také pole provedených SQL dotazů, tedy dotazů, které úspěšně provedly a pozměnily databázi. Tyto SQL dotazy se mohou ukládat například do souboru a v případě poruchy databáze se obnoví data ze zálohy a dodatečně se spustí uložené SQL dotazy ze souboru, které doplní a upraví data, aby došlo k co nejmenší ztrátě dat. Pro přesuny indexů opět existují obdobné metody jako u průchodu chybových hlášení. Názvy těchto metod jsou: `nextSQL`, `prevSQL`, `firstSQL`, `lastSQL` a `resetSQL`. Funkce těchto metod odpovídá metodám pro chybová hlášení.

5.3 Přihlašování k úkolům na stránce MediaWiki

Úkoly, ke kterým se studenti přihlašují, jsou uloženy na MediaWiki stránce. Pro získávání dat z MediaWiki stránky a následné ukládání dat je potřeba komunikovat se systémem MediaWiki. Jelikož Ing. Aleš Vavřínek ve své diplomové práci řešil i spojení s MediaWiki, tak byly v této bakalářské práci k tomuto účelu využity jeho funkce. Názvy funkcí jsou `get_content_page` a `update_stranky_projektu`. Více informací o těchto funkcích naleznete v dokumentaci [22]. Díky tomu, že byly využity již hotové funkce, nebyl zanášen do systému NLPIS duplicitní kód. Do této implementační části spadají dvě třídy. Třída `taskWiki`, která je určena pouze pro čtení dat o úkolech pro nové studenty, a třída `taskNLP`, která zpracovává stránku v MediaWiki a aktualizuje data v databázi.

5.3.1 Třída taskWiki

Třída `taskWiki` slouží pouze ke čtení dat z databáze. Smyslem této třídy je vytvořit objekt, který komunikuje s databází, a pro získání požadovaných dat již stačí komunikovat s touto třídou. Odpadá tedy postupné dotazování na databázi a díky tomu se sníží zátěž na ni kladená. Data z databáze jsou načtena do vnitřní paměťové struktury třídy `taskWiki`, která je zobrazena na obrázku č. 4.6.

Tato datová struktura se skládá z pole, které obsahuje data pro jednotlivé wiki úkoly. Každý z těchto wiki úkolů obsahuje ještě dvě nezávislá pole, ve kterých jsou uložení řešitelé přihlášení k projektu a projekty, které z tohoto wiki úkolu vznikly. Každé z těchto polí vyžaduje určitý index, který bude posouván a díky němuž budou čtena požadovaná data.

Indexy mají názvy `taskIndex`, `solverIndex`, `projectIndex` a každý může být posouván těmito metodami:

- Index bude resetován, tedy bude nastaven na hodnotu -1 (`resetTask`, `resetSolver`, `resetTaskProject`).
- Inkrementuje index v daném poli a vrátí pravdivostní hodnotu true, pokud se posun podařil. Pokud se nacházíme na konci pole, tak bude navrácena pravdivostní hodnota false (`nextTask`, `nextSolver`, `nextTaskProject`).
- Dekrementuje index v daném poli. Pokud se nenacházíme na prvním prvku, tak bude navrácena pravdivostní hodnota false. Při úspěšném posunutí indexu bude navrácena pravdivostní hodnota true (`prevTask`, `prevSolver`, `prevTaskProject`).
- Nastaví index v poli na první prvek a vrátí pravdivostní hodnotu true. Pokud je pole prázdné, tak bude vrácena pravdivostní hodnota false (`firstTask`, `firstSolver`, `firstTaskProject`).
- Když není pole prázdné, tak bude index nastaven na poslední prvek v poli a bude navrácena pravdivostní hodnota true. Pokud je pole prázdné, tak bude vrácena hodnota false (`lastTask`, `lastSolver`, `lastTaskProject`).
- Nastaví index na zvolený prvek (`setTask`, `setSolver`, `setTaskProject`)

5.3.2 Čtení dat z databáze

Čtení dat o wiki úkolech se provádí metodou `getTasks`. Ale jelikož je tato metoda víceúčelová, tak pro snadnější použití obsahuje třída `taskWiki` metody `getAllTasks` a `getTaskByID`. Metoda `getAllTasks` má pouze volitelné parametry, které jsou využity pro úpravu

SQL dotazu a slouží k filtraci dat. Metoda `getTasksByID` má pouze jeden povinný parametr a to je číselný identifikátor úkolu, který má být načten z databáze.

V metodě `getTasks` se ze začátku inicializuje objekt na výchozí hodnoty a následně se sestrojí SQL dotaz, který vybere všechna potřebná data pro výpis wiki úkolů. Tento SQL dotaz je podobný dotazu v třídě `task`. Nastávají zde obdobné problémy se stránkováním, protože jeden wiki úkol může být uveden jako více záznamů, a to z důvodu, že v dotazu jsou zároveň vybrány všichni řešitelé k wiki úkolu a všechny projekty, které vznikly z wiki úkolu. Díky tomu jsou vybrána všechna potřebná data jedním SQL dotazem. Kvůli správnému stránkování je v SQL vložen poddotaz, který dle nastaveného filtru a řazení vybere jen zadaný počet identifikátorů wiki úkolů a na na tyto identifikátory jsou poté nabaleny potřebné informace o wiki úkolech. Stejně jako u třídy `task` i zde se vyskytl problém s SQL dotazem, protože MySQL nepodporuje klauzuli Limit v poddotazu. Problém byl tedy vyřešen přidáním dalšího poddotazu, který zastřeší stávající poddotaz, ale nebude již obsahovat klauzuli LIMIT a dotaz je tedy plně funkční. Po provedení SQL dotazu jsou data tříděna dle identifikátorů jednotlivých úkolů a uložena do pole. Zároveň jsou ke každému wiki úkolu v poli naplněna případná data o řešitelích a projektech vytvořených z wiki úkolu. Takto uspořádané pole se poté projde a do vnitřní datové struktury data se uloží všechny informace. Tento průchod je důležitý z důvodu, aby nově vytvořené pole mělo správné indexy (0, 1, 2, ...) pro pohodlnější hlídaní indexů v poli.

5.3.3 Počet prvků v databázi

Stránkování vyžaduje znát celkový počet prvků, které odpovídají zadaným filtrům. Z tohoto počtu je pak vypočten počet případných stránek, na které se data mají rozdělit. Metody pro zjištění počtu prvků existují v třídě `taskWiki` dvě. První je `countAllTasks`, která vrátí počet všech nalezených wiki úkolů, jenž vyhovují zadaným filtrům, a druhá metoda je `countTaskByID`, pomocí které lze zjistit, jestli zadaný wiki úkol v databázi existuje.

SQL dotaz metody `countTask` je odvozen od SQL dotazu metody `getTask`. Jediný rozdíl je, že tento SQL dotaz nevybírá z databáze data, ale pouze konstantní hodnotu 1. Tím je docíleno vyšší rychlosti při zpracování dotazu.

5.3.4 Metody třídy taskWiki

Při rozšiřování informačního systému NLPIS a práci s třídou `taskWiki` nemusí každému vyhovovat postupné posouvání indexů, aby se dostal k požadovaným datům. Z tohoto důvodu poskytuje třída `taskWiki` metodu `getTaskData`, která vrátí pole obsahující data k právě vybranému wiki úkolu. Metodě `getTaskData` lze pomocí parametru předat číslo indexu, ze kterého se mají data přečíst. Tento parametr může být zadán jako kladné číslo, které udává číselný index v poli, nebo jako záporné číslo, které udává index v poli počítaný od konce (hodnota -2 znamená předposlední prvek). V tomto poli jsou zároveň uložena data o řešitelích wiki úkolu a o projektech, které vznikly z wiki úkolu.

Zajímavými metodami, které usnadňují práci s výpisem dat, jsou `getWholeLeaderName` a `getWholeReceiverName`, které vypisují všechny části jména pro vedoucího a pro příjemce zpráv. Volitelným parametrem těchto metod je řetězec, který je použit jako oddělovač mezi jednotlivými částmi jména. Výchozím oddělovačem je znak mezera.

5.3.5 Třída taskNLP

Tato třída je využita pro obsluhu MediaWiki stránky, která obsahuje úkoly pro nové studenty. Tato stránka má určitou gramatiku, která je uvedena v kapitole 4.3.7. Dle této kapitoly je sestrojen konečný automat, jenž je popsán v kapitole 4.3.5. Před samotným zpracováním stránky je nejprve třeba stránku z MediaWiki načíst. K tomuto účelu je využita funkce z diplomové práce pána Ing. Aleše Vavřínka. Jeho funkce `get_content_page` požaduje parametr, který obsahuje název stránky, jenž se má číst, a navrací řetězec získaný ze stránky MediaWiki. Jelikož funkce `get_content_page` neumožňuje nastavit uživatelské jméno a heslo pro přihlášení k MediaWiki a MediaWiki, na které jsou uloženy wiki úkoly, je na jiném serveru než systém MediaWiki, pro který je diplomová práce pána Ing. Aleše Vavřínka zkonstruována, tak je potřeba před zavoláním této funkce přepsat globální proměnné ze souboru `nastaveni.php`, které obsahují uložené uživatelské jméno a heslo. Jedná se o proměnné `$settings['user']` a `$settings['pass']`. Tento načtený řetězec se zpracuje a vygeneruje se řetězec nový, který se poté vloží do stránky MediaWiki. Uložení do MediaWiki stránky se také dělá pomocí funkce pána Ing. Aleše Vavřínka zvané `update_stranky_projektu`, které se jako parametr předá název upravované stránky a obsah, který se má na danou stránku nahrát. Pro úspěšné uložení dat na MediaWiki stránku je potřeba znovu přepsat proměnné obsahující uživatelské jméno a heslo pro přihlášení do systému MediaWiki.

Kapitola 6

Testování

Jednotlivé části bakalářské práce byly testovány různě, protože se týkaly různých systémů a kladly různé požadavky na testování. Test probíhal na nainstalovaných kopiích používaných systémů, tedy systému MediaWiki a informačního systému NLPIS.

6.1 Automatické generování wiki stránek z e-mailů a IS

Testování této části probíhalo zaváděním chyb do stránky MediaWiki a následným generováním stránky pomocí skriptu pána Ing. Aleše Vavřínka. Stránka před zavedením chyby a po zavedení chyby byla následně porovnána a bylo zjištěno, jestli si skript poradil se změnou stránky či nikoliv. Testování probíhalo v kombinaci s generováním hlavičky pro projekty a skupiny a se zákazem generování hlaviček. Testy se skládaly z jednoduchých změn, které se postupně prolínaly a tvořily komplikovanější chyby.

6.2 Správa úkolů v systému NLPIS

Tato část bakalářské práce byla nejvíce náchylná na možné chyby a proto bylo nutné provést důkladné testy. Protože když dojde k nekonzistenci databáze, tak se data velmi špatně opravují a nejjednodušší je provést obnovu dat ze zálohy. Testování bylo rozvrženo na dvě fáze. První fázi testování byla prováděna na mém počítači, kde byla kopie informačního systému NLPIS, do které byly implementovány nově vytvořené třídy. Po úspěšném dokončení testů nastala druhá fáze a to bylo testování přímo v informačním systému NLPIS. Implementace byla nasazena na server, aby byla otestována uživateli. Jelikož uživatelé si nemusí všimnout chybně interpretovaných dat, tak byly vytvořena speciální funkce `checkTasks`, která prohledávala databázi a kontrolovala, jestli jsou data v databázi konzistentní. Kontrola byla zaměřena na kritické části algoritmu pro vypisování úkolů. Jednou z těchto kritických částí je, že pořadí úkolů musí začínat číslem 1 a musí být navyšována bez vynechání čísla. Tedy pořadí 1, 2, 3, 5 je chybné. Další kontrolovanou věcí bylo, jestli úkol obsahuje právě jeden aktuální stav. Tato kontrola databáze byla prováděna jedenkrát denně a spolu s touto kontrolou byla vytvořena záloha databáze, aby při případném obnovení dat došlo k co nejmenší ztrátě dat.

6.2.1 Efektivita

Jelikož byla předělána správa úkolů, tak úkolem bylo také zároveň pozměnit dosavadní stránku pro vypisování úkolů. Tato stránka vykazovala vysokou neefektivnost, co se SQL do-

tazů týče, a to mělo za následek neúnosně dlouhou dobu načítání stránky. V tabulce 6.1 je zobrazena doba před a po použití nové implementace. Jak je vidět, díky přechodu z velkého počtu malých SQL dotazů na jeden velký SQL dotaz došlo k výraznému urychlení načítání stránky, i když došlo ke zvýšení složitosti načítané stránky (z databáze se již kvůli úkolům nenačítá jeden atribut, ale několik atributů). Tyto časové údaje byly naměřeny v prohlížeči FireFox díky doplňku FireBug, který obsahuje panel sít', ve kterém jsou uvedeny časy načítání stránky.

Implementace	**Čas** [s]
Původní	38,74
Třída task	4,06

Tabulka 6.1: Porovnání časů pro generování stránky úkoly

6.2.2 Zabezpečení formuláře

Třída `taskForm` obsluhuje formulář úkolů v systému NLPIS. Jelikož formulář je vykreslen na straně klienta a ze strany klienta také odesílán, tak existuje možnost, že klient si mohl formulář upravit a zasílá jiný formulář, než mu byl původně vygenerován. V rámci testování byl tento formulář upraven a odzkoušen, zdali si třída `taskForm` s tímto problémem poradí. Úprava formuláře byla provedena v prohlížeči FireFox s doplňkem FireBug, který umožňuje měnit načtená HTML a CSS data.

Testované změny formuláře:

- Změna identifikátoru úkolů na úkol, který nepatří upravovanému projektu.
  Kdyby nebyla tato možnost ošetřena, tak by uživatel mohl měnit úkoly, které nesouvisejí s jeho projektem a nemá ani právo je měnit. Tento stav byl testován přímou editací formuláře, kde se testovalo, jestli třída `taskForm` odhalí takto provedenou změnu a vyhodnotí tento formulář jako chybný.

- Přidání značek input do formuláře, které mají stejný název jako input pro zadání úkolu a termín úkolu pouze s právy řešitele.
  Vygenerovaný formulář se liší dle práv, které uživatel má. Pokud má uživatel pouze práva řešitele, tak je mu zamezeno editovat zadání úkolu a termín úkolu, ale kdyby se uživateli podařilo editovat obsah HTML stránky, tak by mohl provést i odeslání těchto dat. Třída `taskForm` obsahuje při ukládání kontrolu, jestli daný uživatel má práva uložit požadovaná data. Jelikož se ale jedná o možnou bezpečnostní chybu, tak bylo třeba tuto možnost řádně otestovat.

- Přidání formuláře, který obsahuje všechny náležité prvky pro změnu úkolů bez práv vedoucího a bez práv řešitele.
  Uživatel, který nemá práva vedoucího ani práva řešitele, nemá zobrazený formulář, ale při odeslání stránky by mohl naplnit odeslaná data pomocí formuláře, který si na stránce vytvořil. Proto bylo potřeba ošetřit i tuto možnost a následně otestovat, jestli to skutečně funguje.

- Vložení hodnot do formuláře, které by mohly provést takzvaný SQL Injection[1].
  SQL Injection se využívá k získávání dat, ke kterým by se uživatel neměl dostat,

[1] Více o SQL injection [17] [3]

nebo k provádění SQL dotazů, které mohou přivést nekonzistenci do databáze nebo databázi poškodit. V třídě `taskForm` je provedena ochrana proti SQL Injection, kterou bylo třeba otestovat. Do formuláře byly zadávány ony řetězce, které by mohly vést k pozměnění SQL dotazu (výhodou je, že známe formát SQL dotazu, tak víme, kde zadat ony řetězce, aby byl SQL dotaz platný a zároveň mohl napáchat škodu).

6.2.3 Přihlašování k úkolům na stránce MediaWiki

Narozdíl od části Automatické generování wiki stránek z e-mailů a IS, zde se jedná pouze o jednu konkrétní stránku. Proto se naskýtalo méně možností, kde se mohla vyskytnout chyba. Testování probíhalo postupným přidáváním elementárních chyb, které měly za následek porušení gramatiky stránky s wiki úkoly. Po úspěšném otestování elementárních chyb u každého z rozdílných celků (sekce, skupina, projekt) se přešlo ke složitějšímu testování, kdy se testovaly kombinace těchto chyb.

Kapitola 7

Závěr

V rámci semestrálního projektu jsem analyzoval zadání a jeho součásti. Analýza obsahovala nastudování diplomové práce pána Ing. Aleše Vavřínka, zpracování požadavků pro správu úkolů v systému NLPIS, prostudování syntaxe systému MediaWiki a rozboru MediaWiki stránky pro přihlášování k vypsaným wiki úkolům.

Navrhl jsem řešení tak, aby byly splněny zadané požadavky. Hlavní částí návrhu byly vytvořené třídy, které pracují s daty pro správu úkolů a pro přihlašování řešitelů k wiki úkolům. K těmto třídám byly navrženy databázové tabulky, které tyto třídy obsluhují. Dále byl navržen konečný automat, který čte data na MediaWiki stránce, kde se přihlašují řešitelé k zadaným úkolům.

V letním semestru jsem již implementoval navržené řešení. Diplomová práce pána Ing. Aleše Vavřínka byla upravena, aby splňovala požadavek na zamezení generování hlaviček pro vybrané projekty a skupiny. Dále byly v této diplomové práci nahrazeny vypisované úkoly, protože došlo ke změně databázové struktury, ve které byly tyto úkoly uloženy. Implementace správy úkolů byla rozdělena do dvou tříd, kdy jedna z tříd slouží pro čtení dat z databáze a druhá slouží pro zápis dat do databáze. Implementace přihlašování k wiki úkolům byla také rozdělena na dvě třídy, kde jedna z tříd slouží pro čtení a druhá pro zápis dat. Pomocí těchto tříd jsou do systému NLPIS přidána upozornění na přihlášení nového řešitele k úkolu a bylo usnadněno přidávání nového řešitele a nového projektu z wiki úkolu.

Testování provedených změn a nových tříd probíhalo formou uživatelského testování, při kterém byly odstraněny drobné chyby. Uživatelské testování tříd probíhalo nejprve na lokální kopii systému NLPIS a poté byly třídy nasazeny do provozu, kde byly otestovány větším počtem uživatelů. Efektivita třídy `task` byla otestována na stránce, kde se vypisují všechny úkoly v systému NLPIS. Byl porovnán čas s původní implementací a s implementací pomocí mé třídy. Čas potřebný k načtení stránky byl snížen o 90 %.

Všechny třídy jsou tvořeny tak, aby mohly být využity k více účelům. Obsahují řadu metod, které nejsou pro aktuální implementaci potřebné. Díky těmto metodám lze libovolně procházet načtenými daty a využít tato data pro budoucí rozšíření. Třída `task` může být využita pro tvorbu Ganttova diagramu, kdy lze využít informace o změnách úkolů a zakreslit je do grafu. Dalším možným rozšířením je přidání diskuzí k jednotlivým úkolům, kde by mohli řešitelé se svým vedoucím konzultovat nalezené problémy nebo neshody.

Literatura

[1] Free On-line Dictionary of Computing: recursive acronym. FOLDOC, 2010, [Online; cit. 2012-05-06].
URL http://foldoc.org/recursive+acronym

[2] Freeman, A.: *Pro jQuery*. Apress, 2012, ISBN 1430240954.

[3] Friedl, S.: SQL Injection Attacks by Example. unixwiz.net, Říjen 2007, [Online; cit. 2012-05-06].
URL http://www.unixwiz.net/techtips/sql-injection.html

[4] Glass, M. K.; Scouarnec, Y. L.; Naramore, E.; aj.: *Beginning PHP, Apache, MySQL Web Development*. Wrox Press, 2004, ISBN 0-7645-5744-0.

[5] Hickson, I.: HTML5. W3C, Březen 2012, [Online; cit. 2012-05-06].
URL http://www.w3.org/TR/html5/

[6] Hégaret, P. L.: Document Object Model Activity Statement. W3C, Leden 2008, [Online; cit. 2012-05-06].
URL http://www.w3.org/DOM/Activity

[7] Hégaret, P. L.; Whitmerm, R.; Wood, L.: Document Object Model (DOM). W3C, Červen 2009, [Online; cit. 2012-05-06].
URL http://www.w3.org/DOM/

[8] jQuery: Team - jQuery Project. jQuery, 2012, [Online; cit. 2012-05-06].
URL http://jquery.org/team

[9] Kennedy, B.; Musciano, C.: *HTML & XHTML: The Definitive Guide, 6th Edition*. O'Reilly Media, 2006, ISBN 0-596-52732-2.

[10] MediaWiki: MediaWiki history. MediaWiki, Březen 2012, [Online; cit. 2012-05-06].
URL http://www.mediawiki.org/wiki/MediaWiki_history

[11] Meyer, E. A.: *CSS: The Definitive Guide, 3rd Edition*. O'Reilly Media, 2006, ISBN 0-596-52733-0.

[12] Mozilla: JavaScript Language Resources. Mozilla, Duben 2012, [Online; cit. 2012-05-06].
URL https://developer.mozilla.org/en/JavaScript_Language_Resources

[13] Oracle Corporation: History of MySQL. Oracle Corporation, 2012, [Online; cit. 2012-05-06].
URL http://dev.mysql.com/doc/refman/4.1/en/history.html

[14] Oracle Corporation: MySQL 5.6 Replication - Enabling the Next Generation of Web & Cloud Services. Oracle Corporation, 2012, [Online; cit. 2012-05-06].
URL
http://dev.mysql.com/tech-resources/articles/mysql-5.6-replication.html

[15] PHP: PHP: Credits. PHP, Květen 2012, [Online; cit. 2012-05-06].
URL http://php.net/credits.php

[16] PHP: PHP: General Information. PHP, Květen 2012, [Online; cit. 2012-05-06].
URL http://php.net/manual/en/faq.general.php

[17] PHP: PHP: SQL Injection. PHP, Květen 2012, [Online; cit. 2012-05-06].
URL http://php.net/manual/en/security.database.sql-injection.php

[18] Raggett, D.; Hors, A. L.; Jacobs, I.: HTML 4.01 Specification. W3C, Prosinec 1999, [Online; cit. 2012-05-06].
URL http://www.w3.org/TR/html4/

[19] Raggett, D.; Lam, J.; Alexander, I.; aj.: *Raggett on HTML 4*. Addison-Wesley, 1998, ISBN 0-201-17805-2.

[20] Ray, E. T.: *Learning XML, 2nd Edition*. O'Reilly Media, 2003, ISBN 0-596-00420-6.

[21] Sambells, J.; Gustafson, A.: *AdvancED DOM Scripting: Dynamic Web Design Techniques*. Apress, 2007, ISBN 1590598563.

[22] Vavřínek, B. A.: *Automatické generování wiki stránek z e-mailů a IS*. Diplomová práce, FIT VUT v Brně, 2011.
URL http://www.fit.vutbr.cz/study/DP/rpfile.php.cs?id=11072&y=0

[23] W3C: Extensible Markup Language (XML). W3C, Leden 2012, [Online; cit. 2012-05-06].
URL http://www.w3.org/XML/

[24] Wikipedia: Håkon Wium Lie. Wikipedia, Leden 2012, [Online; cit. 2012-05-06].
URL http://en.wikipedia.org/wiki/H%C3%A5kon_Wium_Lie

[25] Wilton, P.; McPeak, J.: *Beginning JavaScript, 3rd Edition*. Wrox Press, 2007, ISBN 1-4571-0415-6.

[26] Zawodny, J. D.; Balling, D. J.: *High Performance MySQL*. O'Reilly Media, 2004, ISBN 0-596-00306-4.

Přílohy

Seznam příloh

Příloha A

Obsah CD

- documentation - dokumentace ve fotmátu LaTeX
- doxygen - vygenerovaná doxygen dokumentace
- images - obrázky použité v dokumentaci
- other - ostatní věci k dokumentaci
- source - zdrojové kódy

Příloha B

Diagramy tříd

task

-taskData: array
-groupIndex: int
-changeIndex: int
-taskIndex: int
-stateIndex: int
-history: bool
-states: bool
+NoHistory: int = 0
+History: int = 1
+NoStates: int = 0
+States: int = 2
+nullReplace: string = "-"

+__construct(): void
+getAllTasks(settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): bool
+getTasksByID(id: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): bool
+getTasksByUserProject(user: int, proj: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): bool
+getTasksByUser(user: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): bool
+getTasksByProject(proj: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): bool
-getTasks(idTask: int, idProj: int, idUser: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): bool
+countAllTasks(settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): int
+countTasksByID(id: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): int
+countTasksByUserProject(user: int, proj: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): int
+countTasksByUser(user: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): int
+countTasksByProject(proj: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): int
-countTasks(idTask: int, idProj: int, idUser: int, settings: int = NULL, where: string = NULL, order: string = NULL, start: int = NULL, limit: int = NULL): int
+getTaskData(groupIndex: int = NULL, changeIndex: int = NULL, taskIndex: int = NULL): array
+getStateData(groupIndex: int = NULL, changeIndex: int = NULL, taskIndex: int = NULL, stateIndex: int = NULL): array
+nextGroup(): bool
+prevGroup(): bool
+resetGroup(): void
+firstGroup(): bool
+lastGroup(): bool
+nextChange(): bool
+prevChange(): bool
+resetChange(): void
+firstChange(): bool
+lastChange(): bool
+nextTask(): bool
+prevTask(): bool
+resetTask(): void
+firstTask(): bool
+lastTask(): bool
+nextState(): bool
+prevState(): bool
+resetState(): void
+firstState(): bool
+lastState(): bool
+findProject(projectID: int, start: int = 0): bool
+findSolver(solverID: int, start: int = 0): bool
+findTask(taskID: int, start: int = 0): bool
+groupIndex(): int
+changeIndex(): int
+taskIndex(): int
+stateIndex(): int
+groupCount(): int
+changeCount(groupIndex: int = NULL): int
+taskCount(groupIndex: int = NULL, changeIndex: int = NULL): int
+taskFormCount(groupIndex: int = NULL, changeIndex: int = NULL): int
+stateCount(groupIndex: int = NULL, changeIndex: int = NULL, taskIndex: int = NULL): int
+getProjectID(): int
+getProjectType(): int
+getProjectName(): string
+getProjectState(): int
+getProjectAcronym(): string
+getProjectGroupID(): int
+getProjectGroupName(): string
+getProjectLeaderID(): int
+getProjectLeaderLogin(): string
+getProjectURL(): string
+getProjectActivity(): int
+translateActivity(activity: int = NULL): string
+getProjectUserID(): int
+getProjectUserEmail(): string
+getProjectUserLogin(): string
+getProjectUserPrefix(): string
+getProjectUserName(): string
+getProjectUserSurname(): string
+getProjectWholeUserName(sep: string = " "): string
+getProjectLastPost()
+translateProjectState(state: string = NULL): string
+getChangeDate()
+getTaskID(): int
+getTaskNumber(): int
+getTaskText(): string
+getTaskDate()
+getTaskDeadline()
+getTaskSolverID(): int
+getTaskSolverNamePrefix(): string
+getTaskSolverName(): string
+getTaskSolverSurname(): string
+getTaskWholeSolverName(sep: string = " "): string
+getTaskStateState(taskID: int): string
+getTaskStateCompleted(taskID: int): int
+getStateState(): int
+getStateCompleted(): int
+getStateComment(): string
+getStateAuthID(): int
+getStateAuthPrefix(): string
+getStateAuthName(): string
+getStateAuthSurname(): string
+getStateWholeAuthName(sep: string = " "): string
+getStateSendTime()
+getStateActual(): bool
+translateTaskState(state: int = NULL): string
+setSelectHistory(num: bool): void
+setSelectStates(num: bool): void
-exist(groupIndex: int = NULL, changeIndex: int = NULL, taskIndex: int = NULL, stateIndex: int = NULL): bool
-isChild(version: int, group: int, change: int): bool

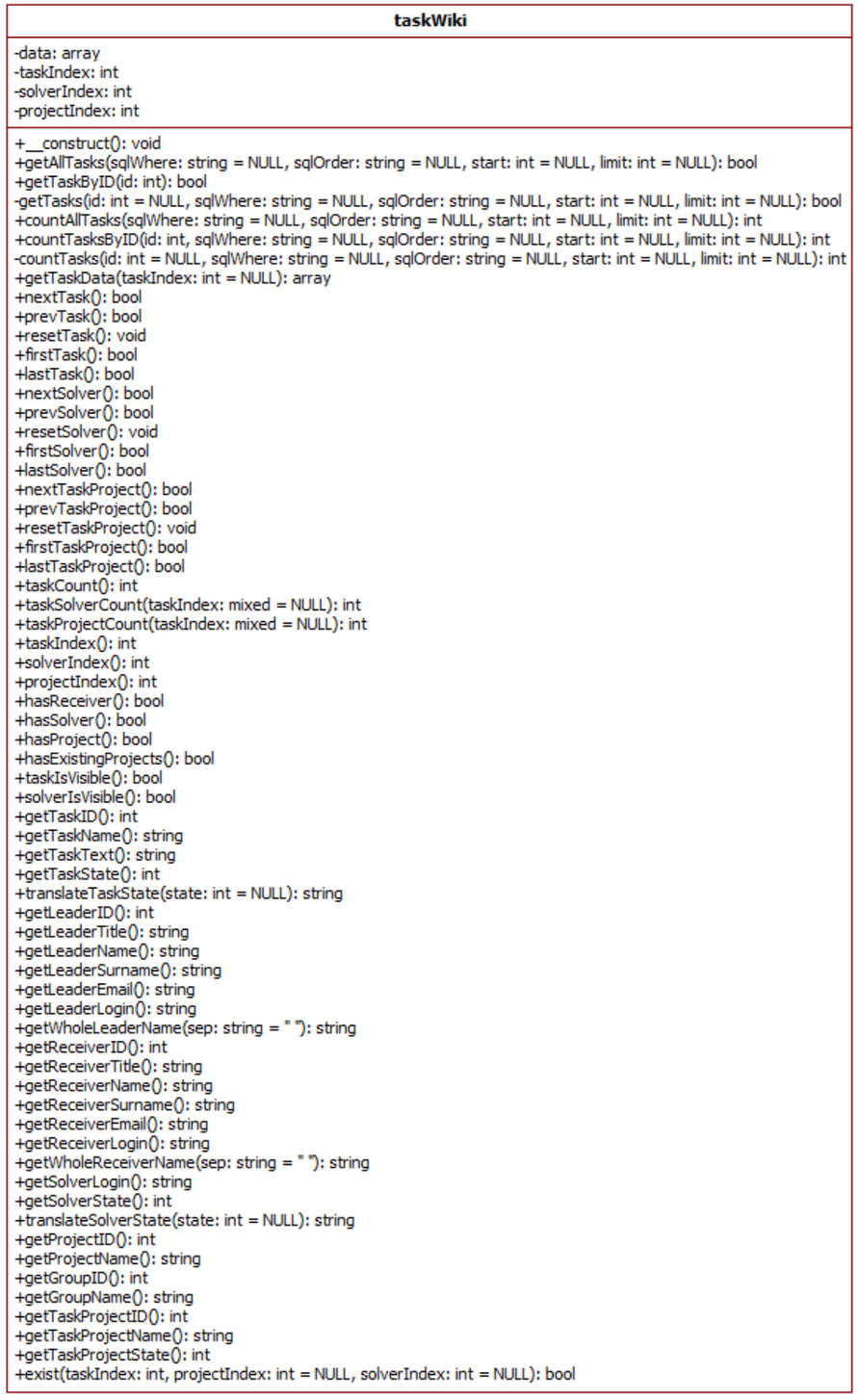
Obrázek B.1: Třída task

taskForm

+sendName: string = "upravProj"
+textName: string = "task"
+stateName: string = "state"
+completedName: string = "completed"
+deadlineName: string = "deadline"
+idName: string = "id"
+commentName: string = "comment"
+SQLCanChangeProject: string = " A%"
-sendNameVar: string
-textNameVar: string
-stateNameVar: string
-completedNameVar: string
-deadlineNameVar: string
-idNameVar: string
-commentNameVar: string
-errArray: array
-errIndex: int
-errSQL: array
-errSQLIndex: int
-SQLArray: array
-SQLIndex: int
-task: task
-projectID: int
-solverID: int
-userID: int
-leaderTemp: bool

+__construct(task, userID: int, sendName = NULL, idName: string = NULL, textName: string = NULL, stateName: string = NULL, completedName: string = NULL, deadlineName: string = NULL, commentName: string = NULL): void
+saveTasks(new: int = 0): bool
-taskOldState(taskID: int, state: int): bool
+getTextName(): string
+getStateName(): string
+getCompletedName(): string
+getDeadlineName(): string
+getSendName(): string
+getIDName(): string
+getCommentName(): string
+getUserID(): int
+setUserID(id: int): void
+setProjectID(id: int): void
+setSolverID(id: int): void
+checkForm()
-addError(text: string): void
+hasError(): bool
+errorCount(): int
+firstError(): bool
+lastError(): bool
+resetError(): void
+nextError(): bool
+prevError(): bool
+readError(): string
+getErrorArray(): array
-addSQLError(text: string): void
+hasSQLError(): bool
+errorSQLCount(): int
+firstSQLError(): bool
+lastSQLError(): bool
+resetSQLError(): void
+nextSQLError(): bool
+prevSQLError(): bool
+readSQLError(): string
+getSQLErrorArray(): array
+hasSQL(): bool
+SQLCount(): int
+firstSQL(): bool
+resetSQL(): void
+nextSQL(): bool
+readSQL(): string
+getSQLArray(): array
+isSend(num: int = NULL): bool
+isEmpty(): bool
+isLeader(permission: string = "", strict: int = 0): bool
+isSolver(): bool
+isChanged(): bool

Obrázek B.2: Třída taskForm



Obrázek B.3: Třída taskWiki

taskNLP

-index: int
-actState: int
-lines: array
-debug: bool
-input: string
-output: string
-searchInput: string
-errArray: array
-errIndex: int
+noRewrite: int = 1
+rewriteWiki: int = 2
+rewriteDB: int = 3
+section: int = 1
+group: int = 2
+project: int = 3
+isHidden: int = 0
+isVisible: int = 1
+isErasable: int = 2
+isPrintable: int = 3
-userName: string = "jmeno"
-userPass: string = "heslo"
-state: array = Array("error"=>0,"unknown"=>1,"notoc"=>2,"sectionID"=>3,"sectionHash"=>4,"sectionTitle"=>5,"sectionText"=>6,"sectionSave"=>7,"groupID"=>8,"groupHash"=>9,"groupTitle"=>10,"groupText"=>11,"groupSave"=>12,"projectHash"=>13,"projectID"=>14,"projectTitle"=>15,"projectSolvers"=>16,"projectBody"=>17,"projectLeader"=>18,"projectProject"=>19,"projectTasks"=>20,"projectSave"=>21,"errorSection"=>22,"eof"=>23)

+__construct(str: string = NULL): void
+load(page: string): bool
+save(page: string): bool
+getInputPage(): string
+getOutputPage(): string
+proccess(debug: int = 1): void
-nextIndex(endState: int = NULL, stripNL: int = 1): bool
-deleteErrorSections(inputArray: array, outputType: int = 0): void
+debug(note: int = 1): string
-addDebug(error: bool, state: int = NULL, index: int = NULL): void
-addError(text: string): void
+hasError(): bool
+errorCount(): int
+firstError(): bool
+resetError(): void
+nextError(): bool
+readError(): bool
+getErrorArray(): array
-compare(a: string, b: string, preRegExp: string = NULL): bool
-compareHash(db: string, page: string, text: string): int
-removeID(array: array, id: int): array
-generateBlock(id: int, type: int, parentID: int): string
-generateSection(sID: int): string
-generateGroup(gID: int): string
-generateProject(pID: int): string
-proccessMissing(missing: array, type: int, parentID: int): string
-proccessMissingProjects(missing: array, type: int, parentID: int): string
-hideBlock(id: int, type: int): int
-hideMissing(missing: array, type: int): bool
-hideMissingBlocks(IDlist: array, type: int): bool
-generateMissing(missing: array, type: int, parentID: int): string
-generateMissingBlocks(IDlist: array, type: int, parentID: int): string
-saveSection(sID: int, contentSection: string, action: int): bool
-updateSection(sID: int, content: string): bool
-insertSection(content: string): bool
-saveProject(pID: int, contentProject: string, action: int): bool
-updateProject(pID: int, content: string): bool
-insertProject(content: string): bool
-deleteProject(pID: int): bool
-getPageSections(): array
-getSectionGroups(sID: int): array
-getGroupProjects(gID: mixed): array
-getProjectState(pID: int): int
-isHidden(pID: int): int
-isVisible(pID: int): bool
-isErasable(pID: int): bool
-isPrintable(pID: int): bool
-updateStates(): void
-hasChanged(pID: int): int

Obrázek B.4: Třída taskNLP